

Lecture Notes in Computer Science

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

2832

Springer

Berlin

Heidelberg

New York

Hong Kong

London

Milan

Paris

Tokyo

Giuseppe Di Battista Uri Zwick (Eds.)

Algorithms – ESA 2003

11th Annual European Symposium
Budapest, Hungary, September 16-19, 2003
Proceedings



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Giuseppe Di Battista
Università degli Studi "Roma Tre", Dipartimento di Informatica e Automazione
via della Vasca Navale 79, 00146 Rome, Italy
E-mail: gdb@dia.uniroma3.it

Uri Zwick
Tel Aviv University, School of Computer Science
Tel Aviv 69978, Israel
E-mail: zwick@cs.tau.ac.il

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie;
detailed bibliographic data is available in the Internet at <<http://dnb.ddb.de>>.

CR Subject Classification (1998): F.2, G.1-2, E.1, F.1.3, I.3.5, C.2.4, E.5

ISSN 0302-9743

ISBN 3-540-20064-9 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin GmbH
Printed on acid-free paper SPIN: 10955604 06/3142 5 4 3 2 1 0

Preface

This volume contains the 66 contributed papers and abstracts of the three invited lectures presented at the 11th Annual European Symposium on Algorithms (ESA 2003), held in Budapest, September 16–19, 2003. The papers in each section of the proceedings are arranged alphabetically. The three distinguished invited speakers were Bernard Chazelle, Roberto Tamassia, and Éva Tardos.

For the second time, ESA had two tracks, with separate program committees, which dealt respectively with:

- ★ The design and mathematical analysis of algorithms (the “Design and Analysis” track);
- ★ Real-world applications, engineering, and experimental analysis of algorithms (the “Engineering and Applications” track).

Previous ESAs were held at Bad Honnef, Germany (1993); Utrecht, The Netherlands (1994); Corfu, Greece (1995); Barcelona, Spain (1996); Graz, Austria (1997); Venice, Italy (1998); Prague, Czech Republic (1999); Saarbrücken, Germany (2000); Århus, Denmark (2001), and Rome, Italy (2002). The predecessor to the Engineering and Applications track of ESA was the annual Workshop on Algorithm Engineering (WAE). Previous WAEs were held in Venice, Italy (1997), Saarbrücken, Germany (1998), London, UK (1999), Saarbrücken, Germany (2000), Århus, Denmark (2001), and Rome, Italy (2002).

The proceedings of the previous ESAs were published as Springer-Verlag’s LNCS volumes 726, 855, 979, 1284, 1461, 1643, 1879, 2161, and 2461. The proceedings of the WAEs from 1999 onwards were published as Springer-Verlag’s LNCS volumes 1668, 1982, and 2141.

Papers were solicited in all areas of algorithmic research, including but not limited to: computational biology, computational finance, computational geometry, databases and information retrieval, external-memory algorithms, graph and network algorithms, graph drawing, machine learning, network design, on-line algorithms, parallel and distributed computing, pattern matching and data compression, quantum computing, randomized algorithms, and symbolic computation. The algorithms could be sequential, distributed, or parallel. Submissions were strongly encouraged in the areas of mathematical programming and operations research, including: approximation algorithms, branch-and-cut algorithms, combinatorial optimization, integer programming, network optimization, polyhedral combinatorics, and semidefinite programming.

Each extended abstract was submitted to one of the two tracks. The extended abstracts were read by at least three referees each, and evaluated on their quality, originality, and relevance to the symposium. The program committees of both tracks met at the Università degli Studi “Roma Tre”, on May 23rd and 24th. The Design and Analysis track selected for presentation 46 out of the 119 submitted abstracts. The Engineering and Applications track selected for presentation 20

out of the 46 submitted abstracts. The program committees of the two tracks consisted of:

Design and Analysis Track

Yair Bartal	(Hebrew University, Jerusalem)
Jean-Daniel Boissonnat	(INRIA Sophia Antipolis)
Moses Charikar	(Princeton University)
Edith Cohen	(AT&T Labs – Research, Florham Park)
Mary Cryan	(University of Leeds)
Hal Gabow	(University of Colorado, Boulder)
Bernd Gärtner	(ETH, Zürich)
Krzysztof Loryś	(University of Wrocław)
Kurt Mehlhorn	(MPI, Saarbrücken)
Theis Rauhe	(ITU, København)
Martin Skutella	(Technische Universität Berlin)
Leen Stougie	(CWI, Amsterdam)
Gábor Tardos	(Rényi Institute, Budapest)
Jens Vygen	(University of Bonn)
Uri Zwick (Chair)	(Tel Aviv University)

Engineering and Applications Track

Giuseppe Di Battista (Chair)	(Roma Tre)
Thomas Erlebach	(ETH, Zürich)
Anja Feldmann	(Technische Universität, München)
Michael Hallett	(McGill)
Marc van Kreveld	(Utrecht)
Piotr Krysta	(MPI, Saarbrücken)
Burkard Monien	(Paderborn)
Guido Proietti	(L'Aquila)
Tomasz Radzik	(King's College London)
Ioannis G. Tollis	(UT Dallas)
Karsten Weihe	(Darmstadt)

ESA 2003 was held along with the third Workshop on Algorithms in Bioinformatics (WABI 2003), a workshop on Algorithmic Methods and Models for Optimization of RailwayS (ATMOS 2003), and the first Workshop on Approximation and Online Algorithms (WAOA 2003) in the context of the combined conference ALGO 2003. The organizing committee of ALGO 2003 consisted of:

János Csirik (Chair)
Csanád Imreh

both from the University of Szeged.

ESA 2003 was sponsored by EATCS (the European Association for Theoretical Computer Science), the Hungarian Academy of Science, the Hungarian National Foundation of Science, and the Institute of Informatics of the University of Szeged. The EATCS sponsorship included an award of EUR 500 for the

authors of the best student paper at ESA 2003. The winners of this prize were Mohammad Mahdian and Martin Pál for their paper *Universal Facility Location*.

Uri Zwick would like to thank Yedidyah Bar-David and Anat Lotan for their assistance in handling the submitted papers and assembling these proceedings.

We hope that this volume offers the reader a representative selection of some of the best current research on algorithms.

July 2003

Giuseppe Di Battista and Uri Zwick

Reviewers

We would like to thank the reviewers for their timely and invaluable contribution.

Dimitris Achlioptas	Graham Cormode	Alexander Hall
Udo Adamy	Péter Csorba	Magnús M. Halldórsson
Pankaj Agarwal	Ovidiu Daescu	Dan Halperin
Steve Alpern	Mark de Berg	Eran Halperin
Sai Anand	Camil Demetrescu	Sariel Har-Peled
Richard Anderson	Olivier Devillers	Jason Hartline
David Applegate	Walter Didimo	Tzvika Hartman
Claudio Arbib	Martin Dyer	Stephan Held
Aaron Archer	Alon Efrat	Michael Hoffmann
Lars Arge	Robert Elsässer	Philip Holman
Georg Baier	Lars Engebretsen	Cor Hurkens
Euripides Bampis	David Eppstein	Thore Husfeldt
Arye Barkan	Péter L. Erdős	Piotr Indyk
Amotz Barnoy	Torsten Fahle	Kamal Jain
Rene Beier	Dror Feitelson	David Johnson
András Benczúr	Rainer Feldmann	Tomasz Jurdziński
Petra Berenbrink	Kaspar Fischer	Juha Kaerkkäinen
Alex Berg	Matthias Fischer	Kostantinos Kakoulis
Marcin Bieńkowski	Aleksei Fishkin	Przemysław Kanarek
Philip Bille	Tamás Fleiner	Lutz Kettner
Markus Bläser	Lisa Fleischer	Sanjeev Khanna
Avrim Blum	Luca Forlizzi	Samir Khuller
Hans Bodlaender	Alan Frieze	Marcin Kik
Ulrich Brenner	Stefan Funke	Georg Klierer
Gerth Stølting Brodal	Martin Gairing	Ekkehard Köhler
Adam Buchsbaum	Rajiv Gandhi	Jochen Könnemann
Stefan Burkhardt	Maciej Gebala	Guy Kortsarz
John Byers	Bert Gerards	Mirosław Korzeniowski
Gruia Calinescu	Joachim Giesen	Sven Oliver Krumke
Hana Chockler	Roberto Grossi	Daniel Kucner
David Cohen-Steiner	Sven Grothklops	Stefano Leonardi

Mariusz Lewicki	Boaz Pratt-Shamir	Maxim Sviridenko
Giuseppe Liotta	Kirk Pruhs	Tibor Szabó
Francesco Lo Presti	Mathieu Raffinot	Tami Tamir
Ulf Lorenz	Paweł Rajba	Éva Tardos
Thomas Lücking	Rajeev Raman	Monique Teillaud
Matthias Mann	April Rasala	Jan Arne Telle
Conardo Martinez	Dieter Rautenbach	Laura Toma
Jens Maßberg	John Reif	Marc Uetz
Giovanna Melideo	Yossi Richter	R.N. Uma
Manor Mendel	Manuel Rode	Jan van den Heuvel
Michael Merritt	Günter Rote	Frank van der Stappen
Ulrich Meyer	Tim Roughgarden	René van Oostrum
Matus Mihalak	Tibor Rózański	Remco C. Veltkamp
Joseph Mitchell	Peter Sanders	Luca Vismara
Haiko Müller	Stefan Schamberger	Berthold Vöcking
Dirk Müller	Anna Schulze	Tjark Vredeveld
M. Müller-Hannemann	Ingo Schurr	Danica Vukadinovic
S. Muthukrishnan	Micha Sharir	Peng-Jun Wan
Enrico Nardelli	Nir Shavit	Ron Wein
Gaia Nicosia	David Shmoys	Emo Welzl
Yoshio Okamoto	Riccardo Silvestri	Jürgen Werber
Rasmus Pagh	Gábor Simonyi	Gábor Wiener
Katarzyna Paluch	Naveen Sivadasan	Gerhard Woeginger
Victor Pan	San Skulrattanakulchai	Marcin Wrzeszcz
Maurizio Patrignani	Shakhar Smorodinsky	Shengxiang Yang
Rudi Pendavingh	Bettina Speckmann	Neal Young
Christian N.S. Pedersen	Venkatesh Srinivasan	Mariette Yvinec
Paolo Penna	Grzegorz Stachowiak	Martin Zachariasen
Sven Peyer	Stamatis Stefanakos	Paweł Zalewski
Marek Piotrów	Nicolas Stier	An Zhu
Maurizio Pizzonia	Miloš Stojaković	Grażyna Zwoźniak
Thomas Plachetka	Frederik Stork	
Tobias Polzin	Torsten Suel	

Table of Contents

Invited Lectures

Sublinear Computing	1
<i>Bernard Chazelle</i>	
Authenticated Data Structures	2
<i>Roberto Tamassia</i>	
Approximation Algorithms and Network Games	6
<i>Éva Tardos</i>	

Contributed Papers: Design and Analysis Track

I/O-Efficient Structures for Orthogonal Range-Max and Stabbing-Max Queries	7
<i>Pankaj K. Agarwal, Lars Arge, Jun Yang, Ke Yi</i>	
Line System Design and a Generalized Coloring Problem	19
<i>Mansoor Alicherry, Randeep Bhatia</i>	
Lagrangian Relaxation for the k-Median Problem: New Insights and Continuity Properties	31
<i>Aaron Archer, Ranjithkumar Rajagopalan, David B. Shmoys</i>	
Scheduling for Flow-Time with Admission Control	43
<i>Nikhil Bansal, Avrim Blum, Shuchi Chawla, Kedar Dhamdhere</i>	
On Approximating a Geometric Prize-Collecting Traveling Salesman Problem with Time Windows	55
<i>Reuven Bar-Yehuda, Guy Even, Shimon (Moni) Shahar</i>	
Semi-clairvoyant Scheduling	67
<i>Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Kirk Pruhs</i>	
Algorithms for Graph Rigidity and Scene Analysis	78
<i>Alex R. Berg, Tibor Jordán</i>	
Optimal Dynamic Video-on-Demand Using Adaptive Broadcasting	90
<i>Therese Biedl, Erik D. Demaine, Alexander Golynski, Joseph D. Horton, Alejandro López-Ortiz, Guillaume Poirier, Claude-Guy Quimper</i>	

Multi-player and Multi-round Auctions with Severely Bounded Communication	102
<i>Liad Blumrosen, Noam Nisan, Ilya Segal</i>	
Network Lifetime and Power Assignment in ad hoc Wireless Networks ...	114
<i>Gruia Calinescu, Sanjiv Kapoor, Alexander Olshevsky, Alexander Zelikovsky</i>	
Disjoint Unit Spheres Admit at Most Two Line Transversals	127
<i>Otfried Cheong, Xavier Goaoc, Hyeon-Suk Na</i>	
An Optimal Algorithm for the Maximum-Density Segment Problem	136
<i>Kai-min Chung, Hsueh-I Lu</i>	
Estimating Dominance Norms of Multiple Data Streams	148
<i>Graham Cormode, S. Muthukrishnan</i>	
Smoothed Motion Complexity	161
<i>Valentina Damerow, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, Christian Sohler</i>	
Kinetic Dictionaries: How to Shoot a Moving Target	172
<i>Mark de Berg</i>	
Deterministic Rendezvous in Graphs	184
<i>Anders Dessmark, Pierre Fraigniaud, Andrzej Pelc</i>	
Fast Integer Programming in Fixed Dimension	196
<i>Friedrich Eisenbrand</i>	
Correlation Clustering – Minimizing Disagreements on Arbitrary Weighted Graphs	208
<i>Dotan Emanuel, Amos Fiat</i>	
Dominating Sets and Local Treewidth	221
<i>Fedor V. Fomin, Dimitrios M. Thilikos</i>	
Approximating Energy Efficient Paths in Wireless Multi-hop Networks	230
<i>Stefan Funke, Domagoj Matijevic, Peter Sanders</i>	
Bandwidth Maximization in Multicasting	242
<i>Naveen Garg, Rohit Khandekar, Keshav Kunal, Vinayaka Pandit</i>	
Optimal Distance Labeling for Interval and Circular-Arc Graphs	254
<i>Cyril Gavoille, Christophe Paul</i>	
Improved Approximation of the Stable Marriage Problem	266
<i>Magnús M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, Hiroki Yanagisawa</i>	

Fast Algorithms for Computing the Smallest k -Enclosing Disc	278
<i>Sariel Har-Peled, Soham Mazumdar</i>	
The Minimum Generalized Vertex Cover Problem	289
<i>Refael Hassin, Asaf Levin</i>	
An Approximation Algorithm for MAX-2-SAT with Cardinality Constraint	301
<i>Thomas Hofmeister</i>	
On-Demand Broadcasting Under Deadline	313
<i>Bala Kalyanasundaram, Mahe Velauthapillai</i>	
Improved Bounds for Finger Search on a RAM	325
<i>Alexis Kaporis, Christos Makris, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas, Christos Zaroliagis</i>	
The Voronoi Diagram of Planar Convex Objects	337
<i>Menelaos I. Karavelas, Mariette Yvinec</i>	
Buffer Overflows of Merging Streams	349
<i>Alex Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir</i>	
Improved Competitive Guarantees for QoS Buffering	361
<i>Alex Kesselman, Yishay Mansour, Rob van Stee</i>	
On Generalized Gossiping and Broadcasting	373
<i>Samir Khuller, Yoo-Ah Kim, Yung-Chun (Justin) Wan</i>	
Approximating the Achromatic Number Problem on Bipartite Graphs . . .	385
<i>Guy Kortsarz, Sunil Shende</i>	
Adversary Immune Leader Election in ad hoc Radio Networks	397
<i>Miroslaw Kutylowski, Wojciech Rutkowski</i>	
Universal Facility Location	409
<i>Mohammad Mahdian, Martin Pál</i>	
A Method for Creating Near-Optimal Instances of a Certified Write-All Algorithm	422
<i>Grzegorz Malewicz</i>	
I/O-Efficient Undirected Shortest Paths	434
<i>Ulrich Meyer, Norbert Zeh</i>	
On the Complexity of Approximating TSP with Neighborhoods and Related Problems	446
<i>Shmuel Safra, Oded Schwartz</i>	

A Lower Bound for Cake Cutting	459
<i>Jiří Sgall, Gerhard J. Woeginger</i>	
Ray Shooting and Stone Throwing	470
<i>Micha Sharir, Hayim Shaul</i>	
Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs	482
<i>Aleksandrs Slivkins</i>	
Binary Space Partition for Orthogonal Fat Rectangles	494
<i>Csaba D. Tóth</i>	
Sequencing by Hybridization in Few Rounds.....	506
<i>Dekel Tsur</i>	
Efficient Algorithms for the Ring Loading Problem with Demand Splitting	517
<i>Bing-Feng Wang, Yong-Hsian Hsieh, Li-Pu Yeh</i>	
Seventeen Lines and One-Hundred-and-One Points	527
<i>Gerhard J. Woeginger</i>	
Jacobi Curves: Computing the Exact Topology of Arrangements of Non-singular Algebraic Curves	532
<i>Nicola Wolpert</i>	

Contributed Papers: Engineering and Application Track

Streaming Geometric Optimization Using Graphics Hardware	544
<i>Pankaj K. Agarwal, Shankar Krishnan, Nabil H. Mustafa, Suresh Venkatasubramanian</i>	
An Efficient Implementation of a Quasi-polynomial Algorithm for Generating Hypergraph Transversals	556
<i>E. Boros, K. Elbassioni, V. Gurvich, Leonid Khachiyan</i>	
Experiments on Graph Clustering Algorithms	568
<i>Ulrik Brandes, Marco Gaertler, Dorothea Wagner</i>	
More Reliable Protein NMR Peak Assignment via Improved 2-Interval Scheduling	580
<i>Zhi-Zhong Chen, Tao Jiang, Guohui Lin, Romeo Rizzi, Jianjun Wen, Dong Xu, Ying Xu</i>	
The Minimum Shift Design Problem: Theory and Practice	593
<i>Luca Di Gaspero, Johannes Gärtner, Guy Kortsarz, Nysret Musliu, Andrea Schaerf, Wolfgang Slany</i>	

Loglog Counting of Large Cardinalities	605
<i>Marianne Durand, Philippe Flajolet</i>	
Packing a Trunk	618
<i>Friedrich Eisenbrand, Stefan Funke, Joachim Reichel, Elmar Schömer</i>	
Fast Smallest-Enclosing-Ball Computation in High Dimensions	630
<i>Kaspar Fischer, Bernd Gärtner, Martin Kutz</i>	
Automated Generation of Search Tree Algorithms for Graph Modification Problems	642
<i>Jens Gramm, Jiong Guo, Falk Hüffner, Rolf Niedermeier</i>	
Boolean Operations on 3D Selective Nef Complexes: Data Structure, Algorithms, and Implementation	654
<i>Miguel Granados, Peter Hachenberger, Susan Hert, Lutz Kettner, Kurt Mehlhorn, Michael Seel</i>	
Fleet Assignment with Connection Dependent Ground Times	667
<i>Sven Grothklags</i>	
A Practical Minimum Spanning Tree Algorithm Using the Cycle Property	679
<i>Irit Katriel, Peter Sanders, Jesper Larsson Träff</i>	
The Fractional Prize-Collecting Steiner Tree Problem on Trees	691
<i>Gunnar W. Klau, Ivana Ljubić, Petra Mutzel, Ulrich Pferschy, René Weiskircher</i>	
Algorithms and Experiments for the Webgraph	703
<i>Luigi Laura, Stefano Leonardi, Stefano Millozzi, Ulrich Meyer, Jop F. Sibeyn</i>	
Finding Short Integral Cycle Bases for Cyclic Timetabling	715
<i>Christian Liebchen</i>	
Slack Optimization of Timing-Critical Nets	727
<i>Matthias Müller-Hannemann, Ute Zimmermann</i>	
Multisampling: A New Approach to Uniform Sampling and Approximate Counting	740
<i>Piotr Sankowski</i>	
Multicommodity Flow Approximation Used for Exact Graph Partitioning	752
<i>Meinolf Sellmann, Norbert Sensen, Larissa Timajev</i>	

A Linear Time Heuristic for the Branch-Decomposition of Planar
Graphs 765
 Hisao Tamaki

Geometric Speed-Up Techniques for Finding Shortest Paths in Large
Sparse Graphs 776
 Dorothea Wagner, Thomas Willhalm

Author Index 789

Sublinear Computing

Bernard Chazelle

Department of Computer Science, Princeton University
`chazelle@cs.princeton.edu`

Abstract. Denied preprocessing and limited to a tiny fraction of the input, what can a computer hope to do? Surprisingly much, it turns out. A blizzard of recent results in property testing, streaming, and sublinear approximation algorithms have shown that, for a large class of problems, all but a vanishing fraction of the input data is essentially unnecessary. While grounding the discussion on a few specific examples, I will review some of the basic principles at play behind this “sublinearity” phenomenon.

Authenticated Data Structures

Roberto Tamassia

Department of Computer Science
Brown University
Providence, RI 02912-1910, USA
`rt@cs.brown.edu`
<http://www.cs.brown.edu/~rt/>

Abstract. Authenticated data structures are a model of computation where untrusted responders answer queries on a data structure on behalf of a trusted source and provide a proof of the validity of the answer to the user. We present a survey of techniques for designing authenticated data structures and overview their computational efficiency. We also discuss implementation issues and practical applications.

1 Introduction

Data replication applications achieve computational efficiency by caching data at servers near users but present a major security challenge. Namely, how can a user verify that the data items replicated at a server are the same as the original ones from the data source? For example, stock quotes from the New York Stock Exchange are distributed to brokerages and financial portals that provide quote services to their customers. An investor that gets a stock quote from a web site would like to have a secure and efficient mechanism to verify that this quote is identical to the one that would be obtained by querying directly the New York Stock Exchange.

A simple mechanism to achieve the authentication of replicated data consists of having the source digitally sign each data item and replicating the signatures in addition to the data items themselves. However, when data evolves rapidly over time, as is the case for the stock quote application, this solution is inefficient.

Authenticated data structures are a model of computation where an untrusted responder answer queries on a data structure on behalf of a trusted source and provides a proof of the validity of the answer to the user.

In this paper, we present a survey of techniques for designing authenticated data structures and overview bounds on their computational efficiency. We also discuss implementation issues and practical applications.

2 Model

The *authenticated data structure* model involves a structured collection S of objects (e.g., a set or a graph) and three parties: the *source*, the *responder*, and the *user*. A repertoire of *query operations* and optional *update operations* are assumed to be defined over S . The role of each party is as follows:

- The *source* holds the original version of S . Whenever an update is performed on S , the source produces *structure authentication information*, which consists of a signed time-stamped statement about the current version of S .
- The *responder* maintains a copy of S . It interacts with the source by receiving from the source the updates performed on S together with the associated structure authentication information. The responder also interacts with the user by answering queries on S posed by the user. In addition to the answer to a query, the responder returns *answer authentication information*, which consists of (i) the latest structure authentication information issued by the source; and (ii) a *proof* of the answer.
- The *user* poses queries on S , but instead of contacting the source directly, it contacts the responder. However, the user trusts the source and not the responder about S . Hence, it verifies the answer from the responder using the associated answer authentication information.

The data structures used by the source and the responder to store collection S , together with the algorithms for queries, updates, and verifications executed by the various parties, form what is called an *authenticated data structure*.

In a practical deployment of an authenticated data structure, there would be several geographically distributed responders. Such a distribution scheme reduces latency, allows for load balancing, and reduces the risk of denial-of-service attacks. Scalability is achieved by increasing the number of responders, which do not require physical security since they are not trusted parties.

3 Overview of Authenticated Data Structures

Throughout this section, we denote with n the size of the collection S maintained by an authenticated data structure.

Early work on authenticated data structures was motivated by the *certificate revocation* problem in public key infrastructure and focused on *authenticated dictionaries*, on which membership queries are performed.

The *hash tree* scheme introduced by Merkle [17,18] can be used to implement a static authenticated dictionary. A hash tree T for a set S stores cryptographic hashes of the elements of S at the leaves of T and a value at each internal node, which is the result of computing a cryptographic hash function on the values of its children. The hash tree uses linear space and has $O(\log n)$ proof size, query time and verification time. A dynamic authenticated dictionary based on hash trees that achieves $O(\log n)$ update time is described in [19]. A dynamic authenticated dictionary that uses a hierarchical hashing technique over skip lists is presented in [9]. This data structure also achieves $O(\log n)$ proof size, query time, update time and verification time. Other schemes based on variations of hash trees have been proposed in [2,6,13].

A detailed analysis of the efficiency of authenticated dictionary schemes based on hierarchical cryptographic hashing is conducted in [22], where precise measures of the computational overhead due to authentication are introduced. Using

this model, lower bounds on the authentication cost are given, existing authentication schemes are analyzed and a new authentication scheme is presented that achieve performance very close to the theoretical optimal.

An alternative approach to the design of authenticated dictionary, based on the *RSA accumulator*, is presented in [10]. This technique achieves constant proof size and verification time and provides a tradeoff between the query and update times. For example, one can achieve $O(\sqrt{n})$ query time and update time.

In [1], the notion of a *persistent authenticated dictionary* is introduced, where the user can issue historical queries of the type “was element e in set S at time t ”.

A first step towards the design of more general authenticated data structures (beyond dictionaries) is made in [5] with the authentication of relational database operations and multidimensional orthogonal range queries.

In [16], a general method for designing authenticated data structures using hierarchical hashing over a search graph is presented. This technique is applied to the design of static authenticated data structures for pattern matching in tries and for orthogonal range searching in a multidimensional set of points.

Efficient authenticated data structures supporting a variety of fundamental search problems on graphs (e.g., path queries and biconnectivity queries) and geometric objects (e.g., point location queries and segment intersection queries) are presented in [12]. This paper also provides a general technique for authenticating data structures that follow the *fractional cascading* paradigm.

The software architecture and implementation of an authenticated dictionary based on skip lists is presented in [11]. A distributed system realizing an authenticated dictionary, is described in [7]. This paper also provides an empirical analysis of the performance of the system in various deployment scenarios. The authentication of distributed data using web services and XML signatures is investigated in [20]. *Prooflets*, a scalable architecture for authenticating web content based on authenticated dictionaries, are introduced in [21].

Work related to authenticated data structures includes [3,4,8,14,15].

Acknowledgements. I would like to thank Michael Goodrich for his research collaboration on authenticated data structures. This work was supported in part by NSF Grant CCR-0098068.

References

1. A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. In *Proc. Information Security Conference (ISC 2001)*, volume 2200 of *LNCS*, pages 379–393. Springer-Verlag, 2001.
2. A. Buldas, P. Laud, and H. Lipmaa. Accountable certificate management using undeniable attestations. In *ACM Conference on Computer and Communications Security*, pages 9–18. ACM Press, 2000.
3. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proc. CRYPTO*, 2002.

4. P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. In *Proc. ACM Conference on Computer and Communications Security*, 2001.
5. P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.
6. I. Gassko, P. S. Gemmell, and P. MacKenzie. Efficient and fresh certification. In *Int. Workshop on Practice and Theory in Public Key Cryptography (PKC '2000)*, volume 1751 of *LNCS*, pages 342–353. Springer-Verlag, 2000.
7. M. T. Goodrich, J. Lentini, M. Shin, R. Tamassia, and R. Cohen. Design and implementation of a distributed authenticated dictionary and its applications. Technical report, Center for Geometric Computing, Brown University, 2002. <http://www.cs.brown.edu/cgc/stms/papers/stms.pdf>.
8. M. T. Goodrich, M. Shin, R. Tamassia, and W. H. Winsborough. Authenticated dictionaries for fresh attribute credentials. In *Proc. Trust Management Conference*, volume 2692 of *LNCS*, pages 332–347. Springer, 2003.
9. M. T. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical report, Johns Hopkins Information Security Institute, 2000. <http://www.cs.brown.edu/cgc/stms/papers/hashskip.pdf>.
10. M. T. Goodrich, R. Tamassia, and J. Hasic. An efficient dynamic and distributed cryptographic accumulator. In *Proc. Int. Security Conference (ISC 2002)*, volume 2433 of *LNCS*, pages 372–388. Springer-Verlag, 2002.
11. M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. 2001 DARPA Information Survivability Conference and Exposition*, volume 2, pages 68–82, 2001.
12. M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Proc. RSA Conference—Cryptographers’Track*, pages 295–313. Springer, LNCS 2612, 2003.
13. P. C. Kocher. On certificate revocation and validation. In *Proc. Int. Conf. on Financial Cryptography*, volume 1465 of *LNCS*. Springer-Verlag, 1998.
14. P. Maniatis and M. Baker. Enabling the archival storage of signed documents. In *Proc. USENIX Conf. on File and Storage Technologies (FAST 2002)*, Monterey, CA, USA, 2002.
15. P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *Proc. USENIX Security Symposium*, 2002.
16. C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. A general model for authentic data publication, 2001. <http://www.cs.ucdavis.edu/~devanbu/files/model-paper.pdf>.
17. R. C. Merkle. Protocols for public key cryptosystems. In *Proc. Symp. on Security and Privacy*, pages 122–134. IEEE Computer Society Press, 1980.
18. R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO ’89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1990.
19. M. Naor and K. Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.
20. D. J. Polivy and R. Tamassia. Authenticating distributed data using Web services and XML signatures. In *Proc. ACM Workshop on XML Security*, 2002.
21. M. Shin, C. Straub, R. Tamassia, and D. J. Polivy. Authenticating Web content with prooflets. Technical report, Center for Geometric Computing, Brown University, 2002. <http://www.cs.brown.edu/cgc/stms/papers/prooflets.pdf>.
22. R. Tamassia and N. Triandopoulos. On the cost of authenticated data structures. Technical report, Center for Geometric Computing, Brown University, 2003. <http://www.cs.brown.edu/cgc/stms/papers/costauth.pdf>.

Approximation Algorithms and Network Games

Éva Tardos*

Department of Computer Science
Cornell University
Ithaca, NY, 14853
eva@cs.cornell.edu

Information and computer systems involve the interaction of multiple participants with diverse goals and interests, such as servers, routers, etc., each controlled by different parties. The future of much of the technology we develop, depends on our ability to ensure that participants cooperate despite their diverse goals and interests. In such settings the traditional approach of algorithm design is not appropriate: there is no single entity that has the information or the power to run such an algorithm. While centralized algorithms cannot be used directly in environments with selfish agents, there are very strong ties with certain algorithmic techniques, and some of the central questions in this area of algorithmic game theory.

In this talk we will approach some of the traditional algorithmic questions in networks from the perspective of game theory. Each participant in an algorithm is viewed as a player in a noncooperative game, where each player acts to selfishly optimize his or her own objective function. The talk will focus on understanding the quality of the selfish outcomes. Selfishness often leads to inefficient outcomes, as is well known by the classical Prisoner's Dilemma. In this talk we will review some recent results on *quantifying* this inefficiency by comparing the outcomes of selfishness to the "best possible" outcome.

We will illustrate the issues via two natural network games: a flow (routing) game, and a network design game. In the network routing problem, the latency of each edge is a monotone function of the flow on the edge. We assume that each agent routes his traffic using the minimum-latency path from his source to his destination, given the link congestion caused by the rest of the network users. We evaluate the outcome by the average user latency obtained. It is counter-intuitive, but not hard to see, that each user minimizing his own latency, may not lead to an efficient overall system.

The network design game we consider is a simple, first model of developing and maintaining networks, such as the Internet, by a large number of selfish agents. Each player has a set of terminals, and the goal of each player is to pay as little as possible, while making sure that his own set of terminals is connected in the resulting graph. In the centralized setting this is known as the generalized Steiner tree problem. We will study the Nash equilibria of a related noncooperative game.

The talk will be based on joint work with Elliott Anshelevich, Anirban Dasgupta, Henry Lin, Tim Roughgarden, and Tom Wexler.

* Research supported in part by ONR grant N00014-98-1-0589.

I/O-Efficient Structures for Orthogonal Range-Max and Stabbing-Max Queries

Pankaj K. Agarwal*, Lars Arge**, Jun Yang, and Ke Yi

Department of Computer Science
Duke University, Durham, NC 27708, USA
{`pankaj,large,junyang,yike`}@cs.duke.edu

Abstract. We develop several linear or near-linear space and I/O-efficient dynamic data structures for orthogonal range-max queries and stabbing-max queries. Given a set of N weighted points in $\mathbb{R}^d$, the range-max problem asks for the maximum-weight point in a query hyper-rectangle. In the dual stabbing-max problem, we are given N weighted hyper-rectangles, and we wish to find the maximum-weight rectangle containing a query point. Our structures improve on previous structures in several important ways.

1 Introduction

Range searching and its variants have been studied extensively in the computational geometry and database communities because of their many important applications. Range-aggregate queries, such as range-count, range-sum, and range-max queries, are some of the most commonly used versions of range searching in database applications. Since many such applications involve massive amounts of data stored in external memory, it is important to consider I/O-efficient structures for fundamental range-searching problems. In this paper, we develop I/O-efficient data structures for answering orthogonal range-max queries, as well as for the dual problem of answering stabbing-max queries.

Problem statement. In the *orthogonal range-max* problem, we are given a set S of N points in $\mathbb{R}^d$ where each point p is assigned a weight $w(p)$, and we wish to build a data structure so that for a query hyper-rectangle Q in $\mathbb{R}^d$, we can compute $\max\{w(p) \mid p \in Q\}$ efficiently. The two-dimensional case is illustrated in Figure 1(a). In the dual *orthogonal stabbing-max* problem, we are given a set S of N hyper-rectangles in $\mathbb{R}^d$ where each rectangle γ is assigned a weight $w(\gamma)$, and want to build a data structure such that for a query point q in $\mathbb{R}^d$, we can

* Supported in part by the National Science Foundation through grants CCR-00-86013, EIA-9972879, EIA-98-70724, EIA-01-31905, and CCR-02-04118, and by a grant from the U.S.–Israel Binational Science Foundation.

** Supported in part by the National Science Foundation through ESS grant EIA-9870734, RI grant EIA-9972879, CAREER grant CCR-9984099, ITR grant EIA-0112849, and U.S.–Germany Cooperative Research Program grant INT-0129182.



Fig. 1. (a) Two-dimensional range queries. (b) Two-dimensional stabbing queries.

compute $\max\{w(\gamma) \mid q \in \gamma\}$ efficiently. The two-dimensional case is illustrated in Figure 1(b). We also consider the dynamic version of the two problems, in which points or hyper-rectangles can be inserted or deleted dynamically. In the following we drop “orthogonal” and often even “max” when referring to the two problems.

We work in the standard external memory model [4]. In this model, the main memory holds M words and each disk access (or I/O) transmits a continuous block of B words between main memory and disk. We assume that $M \geq B^2$ and that any integer less than N , as well as any point or weight, can be stored in a single word. We measure the efficiency of a data structure in terms of the amount of disk space it uses (measured in number of disk blocks) and the number of I/Os required to answer a query or perform an update. We will focus on data structures that use linear or near linear space, that is, use close to $n = \lceil N/B \rceil$ disk blocks.

Related work. Range searching data structures have been studied extensively in the internal memory RAM model of computation. In two dimensions, the best known linear space structure for the range-max problem is by Chazelle [10]. It answers a query in $O(\log^{1+\epsilon} n)$ time in the static case. In the dynamic case, the structure supports queries and updates in $O(\log^3 n \log \log n)$ time. The best known structure for the one-dimensional stabbing-max problem is by Kaplan et al [16]. It uses linear space and supports queries and insertions in $O(\log n)$ time and deletions in $O(\log n \log \log n)$ time. They also discuss how their structure can be extended to higher dimensions. Refer to [10,16] and the survey by Agarwal and Erickson [3] for additional results.

In the external setting, one-dimensional range-max queries can be answered in $O(\log_B n)$ I/Os using a standard B-tree [11,8]. The structure can easily be updated using $O(\log_B n)$ I/Os. For two or higher dimensions, however, no efficient linear-size structure is known; In the two-dimensional case, the *k*dB-tree [18], the cross-tree [14], and the O-tree [15], designed for general range searching, can be modified to answer range-max queries in $O(\sqrt{n})$ I/Os. All of them use linear space. The cross-tree [14] and the O-tree [15] can also be updated in $O(\log_B n)$ I/Os. The CRB-tree [2] designed for range-counting can be modified to support range-max queries in $O(\log_B^2 n)$ I/Os using $O(n \log_B n)$ space.

For the one-dimensional stabbing-max problem, the SB-tree [20] can be used to answer queries in $O(\log_B n)$ I/Os using linear space. Intervals can be inserted into the structure in $O(\log_B n)$ I/Os. However, the SB-tree does not support

deletions. No worst-case efficient structures are known for higher-dimensional stabbing max queries. Refer to recent surveys [5,13] for additional results.

Our results. In this paper we obtain three main results. Our first result is a linear-size structure for answering two-dimensional range-max queries in $O(\log_B^2 n)$ I/Os. This is the first linear-size external memory data structure that can answer such queries in polylogarithmic number of I/Os. Using $O(n \log_B \log_B n)$ space, the structure can be made dynamic so that insertions and deletions can be performed in $O(\log_B^2 n \log_{M/B} \log_B n)$ and $O(\log_B^2 n)$ I/Os amortized, respectively. Refer to Table 1 for a comparison with previous results.

Table 1. Two-dimensional range max query results.

Problem	Space	Query	Insertion	Deletion	Source
2D range max queries (static)	$n \log_B n$	$\log_B^2 n$			[2]
	n	$\log_B^2 n$			New
2D range max queries (dynamic)	n	$\sqrt{n}$	$\log_B n$	$\log_B n$	[14,15]
	$n \log_B \log_B n$	$\log_B^3 n$	$\log_B^2 n \cdot \log_{M/B} \log_B n$	$\log_B^2 n$	New

Our second result is a linear-size dynamic structure for answering one-dimensional stabbing-max queries in $O(\log_B^2 n)$ I/Os. The structure supports both insertions and deletions in $O(\log_B n)$ I/Os. As mentioned, the previously known structure only supported insertions [20].

Our third result is a linear-size structure for answering two-dimensional stabbing max queries in $O(\log_B^4 n)$ I/Os. The structure is an extension of our one-dimensional structure, which also uses our two-dimensional range-max query structure. The structure can be made dynamic with an $O(\log_B^5 n)$ query bound at the cost of a factor of $O(\log_B \log_B n)$ in its size. Insertions and deletions can be performed in $O(\log_B^2 n \log_{M/B} \log_B n)$ and $O(\log_B^2 n)$ I/Os amortized, respectively. Refer to Table 2 for a comparison with previous results.

Table 2. Two-dimensional stabbing max query results.

Problem	Space	Query	Insertion	Deletion	Source
1D stabbing max queries (dynamic)	n	$\log_B n$	$\log_B n$		[20]
	n	$\log_B^2 n$	$\log_B n$	$\log_B n$	New
2D stabbing max queries (static)	n	$\log_B^4 n$			New
2D stabbing max queries (dynamic)	$n \log_B \log_B n$	$\log_B^5 n$	$\log_B^2 n \cdot \log_{M/B} \log_B n$	$\log_B^2 n$	New

Finally, using standard techniques [2,9,12], both our range and stabbing structures can be extended to higher dimensions at the cost of increasing each of the space, query, and update bounds by an $O(\log_B n)$ factor per dimension. Our structures can also be extended and improved in several other ways. For example, our one-dimensional stabbing-max structure can be modified to support general semigroup stabbing queries.

2 Two-Dimensional Range-Max Queries

In this section we describe our structure for the two-dimensional range-max problem. The structure is an external version of a structure by Chazelle [10].

The overall structure. Our structure consists of two parts. The first is simply a B-tree $\mathcal{P}$ on the y -coordinates of the N points in S . It uses $O(n)$ blocks and can be constructed in $O(n \log_B n)$ I/Os. To construct the second part, we first build a base B-tree T with fanout $\sqrt{B}$ on the x -coordinates of S . For each node v of T , let P_v be the sequence of points stored in the subtree rooted at v , sorted by their y -coordinates. Set $N_v = |P_v|$ and $n_v = N_v/B$. With each node v we associate a vertical *slab* σ_v containing P_v . If $v_1, v_2, \dots, v_k$, for $k = \Theta(\sqrt{B})$, are the children of v , then $\sigma_{v_1}, \dots, \sigma_{v_k}$ partition σ_v into k slabs. For $1 \leq i \leq j \leq k$, we refer to the slab $\sigma_v[i : j] = \bigcup_{i=i}^j \sigma_{v_i}$ as a *multi-slab*; there are $O(B)$ multi-slabs at each node of T . Each leaf z of T stores $\Theta(B)$ points in P_z and their weights using $O(1)$ disk blocks. Each internal node v stores two secondary structures $\mathcal{C}_v$ and $\mathcal{M}_v$ requiring $O(n_v/\log_B n)$ blocks each, so that the overall structure uses a total of $O(n)$ blocks. We first describe the functionality of these structures. After describing how to answer a query, we describe their implementation.

For a point $p \in \mathbb{R}^2$, let $rk_v(p)$ denote the rank of p in P_v , i.e., the number of points in P_v whose y -coordinates are not larger than the y -coordinate of p . Given $rk_v(p)$ of a point p , $\mathcal{C}_v$ can be used to determine $rk_{v_i}(p)$ for all children v_i of v using $O(1)$ I/Os. Suppose we know the rank $\rho = rk_v(p)$ of a point $p \in P_v$, we can find the weight of p in $O(\log_B n)$ I/Os using $\mathcal{C}_v$: If v is a leaf, then we examine all the points of P_v and return the weight of the point whose rank is ρ . Otherwise, we use $\mathcal{C}_v$ to find the rank of p in the set P_{v_j} associated with the relevant child v_j , and continue the search recursively in v_j . We call this step the *identification process*.

The other secondary structure $\mathcal{M}_v$ enables us to compute the maximum weight among the points in a given multi-slab and rank range. More precisely, given $1 \leq i \leq j \leq \sqrt{B}$ and $1 \leq \rho_1 \leq \rho_2 \leq N_v$, $\mathcal{M}_v$ can be used to determine in $O(\log_B n)$ I/Os the maximum value in $\{w(p) \mid p \in P_v \cap \sigma_v[i : j] \text{ and } rk_v(p) \in [\rho_1, \rho_2]\}$.

Answering a query. Let $Q = [x_1, x_2] \times [y_1, y_2]$ be a query rectangle. We wish to compute $\max\{w(p) \mid p \in S \cap Q\}$. The overall query procedure is the same as for the CRB-tree [2]. Let z_1 (resp. z_2) be the leaf of T such that σ_{z_1} (resp. σ_{z_2}) contains (x_1, y_1) (resp. (x_2, y_2)). Let ξ be the nearest common ancestor of

z_1 and z_2 . Then $S \cap Q = P_\xi \cap Q$, and therefore it suffices to compute $\max\{w(p) \mid p \in P_\xi \cap Q\}$.

To answer the query we visit the nodes on the paths from the root to z_1 and z_2 in a top-down manner. For any node v on the path from ξ to z_1 (resp. z_2), let l_v (resp. r_v) be the index of the child of v such that $(x_1, y_1) \in \sigma_{l_v}$ (resp. $(x_2, y_2) \in \sigma_{r_v}$), and let Σ_v be the widest multi-slab at v whose x -span is contained in $[x_1, x_2]$. Note that $\Sigma_v = \sigma_v[l_v + 1 : r_v - 1]$ when $v = \xi$ (Figure 2(a)), and that for any other node v on the path from ξ to z_1 (resp. z_2), $\Sigma_v = \sigma_v[l_v + 1 : \sqrt{B}]$ (resp. $\Sigma_v = \sigma_v[1 : r_v - 1]$). At each such node v , we compute the maximum weight of a point in the set $P_v \cap \Sigma_v \cap Q$ in $O(\log_B n)$ I/Os using the secondary structure $\mathcal{C}_v$ and $\mathcal{M}_v$. The answer to Q is then the maximum of the $O(\log_B n)$ obtained weights. We compute the maximum weight in $P_v \cap \Sigma_v \cap Q$ as follows: Let $\rho_v^- = rk_v((x_1, y_1))$ and $\rho_v^+ = rk_v((x_2, y_2))$. If v is the root of T , we compute ρ_v^-, ρ_v^+ in $O(\log_B n)$ I/Os using the B-tree Φ . Otherwise, since we know $\rho_{p(v)}^-, \rho_{p(v)}^+$ at the parent of v , we can compute ρ_v^-, ρ_v^+ in $O(1)$ I/Os using the secondary structure $\mathcal{C}_{p(v)}$ stored at the parent $p(v)$ of v . Once we know ρ_v^-, ρ_v^+ , we find the maximal weight point in $P_v \cap \Sigma_v \cap Q$ in $O(\log_B n)$ I/Os by querying $\mathcal{M}_v$ with the multi-slab Σ_v and the rank interval $[\rho_v^-, \rho_v^+]$. Overall the query procedure uses $O(\log_B n)$ I/Os in $O(\log_B n)$ nodes, for a total of $O(\log_B^2 n)$ I/Os.

Secondary structures. We now describe the secondary structures stored at a node v of T . Since $\mathcal{C}_v$ is the same as a structure used in the CRB-tree [2], we only describe $\mathcal{M}_v$. Recall that $\mathcal{M}_v$ is a data structure of size $O(n_v / \log_B n)$, and for a multi-slab $\sigma_v[i : j]$ and a rank range $[\rho_1, \rho_2]$, it returns the maximum weight of the points in the set $\{p \in \sigma_v[i : j] \cap P_v \mid rk_v(p) \in [\rho_1, \rho_2]\}$. Since the size of $\mathcal{M}_v$ is only $O(n_v / \log_B n)$, it cannot store all the coordinates and weights of the points in P_v explicitly. Instead, we store them in a compressed manner.

Let $\mu = B \log_B n$. We partition P_v into $s = \lceil N_v / \mu \rceil$ chunks $C_1, \dots, C_s$, each (except possibly the last one) of size μ . More precisely, $C_i = \{p \in P_v \mid rk_v(p) \in [(i-1)\mu + 1, i\mu]\}$. Next, we partition each chunk C_i further into *minichunks* of

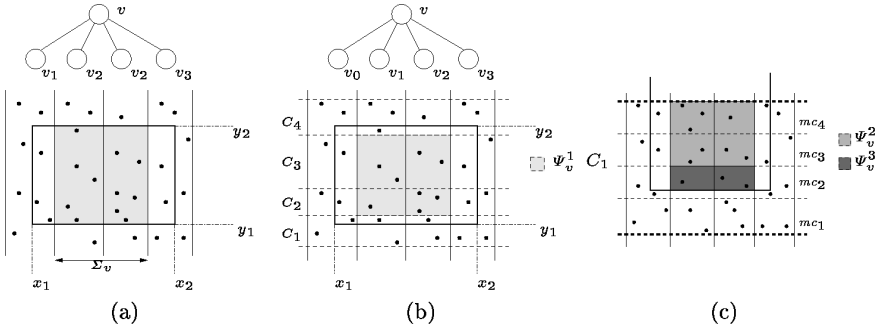


Fig. 2. (a) Answering a query. (b) Finding the max at the chunk level (using Ψ_v^1). (c) Finding the max at the minichunk level (using Ψ_v^2) and within a minichunk (using Ψ_v^3).

size B ; C_i is partitioned into $mc_1, \dots, mc_{\nu_i}$, where $\nu_i = \lceil |C_i|/B \rceil$ and $mc_j \subseteq C_i$ is the sequence of points whose y -coordinates have ranks (within C_i) between $(j-1)B+1$ and jB . We say that a rank range $[\rho_1, \rho_2]$ *spans* a chunk (or a minichunk) X if for all $p \in X$, $rk_v(p) \in [\rho_1, \rho_2]$, and that X *crosses* a rank ρ if there are points $p, q \in X$ such that $rk_v(p) < \rho < rk_v(q)$.

$\mathcal{M}_v$ consists of three data structures Ψ_v^1, Ψ_v^2 , and Ψ_v^3 ; Ψ_v^1 answers max queries at the “chunk level”, Ψ_v^2 answers max queries at the “minichunk level”, and Ψ_v^3 answers max queries within a minichunk. More precisely, let $\sigma_v[i:j]$ be a multi-slab and $[\rho_1, \rho_2]$ be a rank range, if the chunks that are spanned by $[\rho_1, \rho_2]$ are $C_a, \dots, C_b$, then we use Ψ_v^1 to report the maximum weight of the points in $\bigcup_{l=a}^b C_l \cap \sigma_v[i:j]$ (Figure 2(b)). We use Ψ_v^2, Ψ_v^3 to report the the maximum weight of a point in $C_{a-1} \cap \sigma_v[i:j]$, as follows. If $mc_\alpha, \dots, mc_\beta$ are the minichunks of C_{a-1} that are spanned by $[\rho_1, \rho_2]$, then we use Ψ_v^2 to report the maximum weight of the points in $\bigcup_{l=\alpha}^\beta mc_l \cap \sigma_v[i:j]$. Then we use Ψ_v^3 to report the maximum weight of the points that lie in the minichunks that cross ρ_1 (Figure 2(c)). The maximum weight of a point in $C_{b+1} \cap \sigma_v[i:j]$ can be found similarly. Below we describe Ψ_v^1, Ψ_v^2 and Ψ_v^3 in detail and show how they can be used to answer the relevant queries in $O(\log_B n)$ I/Os.

Structure Ψ_v^3 . Ψ_v^3 consists of a small structure $\Psi_v^3[l]$ for each minichunk mc_l , $1 \leq l \leq N_v/B = n_v$. Since we can only use $O(n_v/\log_B n)$ space, we store $\log_B n$ small structures together in $O(1)$ blocks. For each point p in mc_l we store a pair (ξ_p, ω_p) , where ξ_p is the index of the slab containing p , and ω_p is the rank of the weight of p among the points in mc_l (i.e., $\omega_p - 1$ points in mc_l have smaller weights than that of p). Note that $0 \leq \xi_p, \omega_p \leq B$, so we need $O(\log B)$ bits to store this pair. The set $\{(\xi_p, \omega_p) \mid p \in mc_l\}$ is stored in $\Psi_v^3[l]$, sorted in increasing order of $rk_v(p)$'s (their ranks in P_v). $\Psi_v^3[l]$ needs a total of $O(B \log B)$ bits. Therefore $\log_B n$ small structures use $O(B \log B \log_B n) = O(B \log n)$ bits and fit in $O(1)$ disk blocks.

A query on Ψ_v^3 is of the following form: Given a multi-slab $\sigma_v[i:j]$, an interval $[\rho_1, \rho_2]$, and an integer $l \leq n_v$, we wish to return the the maximum weight of a point in the set $\{p \in mc_l \mid p \in \sigma_v[i:j], rk_v(p) \in [\rho_1, \rho_2]\}$. We first load the whole $\Psi_v^3[l]$ structure into memory using $O(1)$ I/Os. Since we know the $rk_v(a)$ of the first point $a \in mc_l$, we can compute in $O(1)$ time the contiguous subsequence of pairs (ξ_p, ω_p) in $\Psi_v^3[l]$ such that $rk_v(p) \in [\rho_1, \rho_2]$. Among these pairs we select the point q for which $i \leq \xi_q \leq j$ (i.e., q lies in the multi-slab $\sigma_v[i:j]$) and ω_q has the largest value (i.e., q has the maximum weight among these points). Since we know $rk_v[q]$, we use the identification process (the $\mathcal{C}_v$ structures) to determine, in $O(\log_B n)$ I/Os, the actual weight of q .

Structure Ψ_v^2 . Similar to Ψ_v^3 , Ψ_v^2 consists of a small structure $\Psi_v^2[k]$ for each chunk C_k . Since there are $N_v/\mu = n_v/\log_B n$ chunks at v , we can use $O(1)$ blocks for each $\Psi_v^2[k]$.

Chunk C_k has $\nu_k \leq \log_B n$ minichunks $mc_1, \dots, mc_{\nu_k}$. For each multi-slab $\sigma_v[i:j]$, we do the following. For each $l \leq \nu_k$, we choose the point of the maximum weight in $\sigma_v[i:j] \cap mc_l$. Let Q_{ij}^k denote the resulting set of points. We

construct a *Cartesian tree* [19] on Q_{ij}^k with their weights as the key. A Cartesian tree on a sequence of weights $w_1, \dots, w_{\nu_k}$ is a binary tree with the maximum weight, say w_k , in the root and with $w_1, \dots, w_{k-1}$ and $w_{k+1}, \dots, w_{\nu_k}$ stored recursively in the left and right subtree, respectively. This way, given a range of minichunks $mc_\alpha, \dots, mc_\beta$ in C_k , the maximal weight in these minichunks is stored in the nearest common ancestor of w_α and w_β . Conceptually, $\Psi_v^2[k]$ consists of such a Cartesian tree for each of the $O(B)$ multi-slabs. However, we do not actually store the weights in a Cartesian tree, but only an encoding of its structure. Thus we can not use it to find the actual maximal weight in a range of minichunks, but only the index of the minichunk containing the maximal weight. It is well known that the structure of a binary tree of size ν_k can be encoded using $O(\nu_k)$ bits. Thus, we use $O(\log_B n)$ bits to encode the Cartesian tree of each of the $O(B)$ multi-slabs, for a total of $O(B \log_B n)$ bits, which again fit in $O(1)$ blocks.

Consider a multi-slab $\sigma_v[i : j]$. To find the maximal weight of the points in the minichunks of a chunk C_k spanned by a rank range $[\rho_1, \rho_2]$, we load the relevant Cartesian tree using $O(1)$ I/Os, and use it to identify the minichunk l containing the maximum-weight point p . Then we use $\Psi_v^3[l]$ to find the rank of p in $O(1)$ I/Os. Finally, we as previously use the identification process to identify the actual weight of p in $O(\log_B n)$ I/Os.

Structure Ψ_v^1 . Ψ_v^1 is a B-tree with fanout $\sqrt{B}$ conceptually built on the $s = n_v / \log_B n$ chunks $C_1, \dots, C_s$. Each leaf of Ψ_v^1 corresponds to $\sqrt{B}$ contiguous chunks, and stores for each of the $\sqrt{B}$ slabs in v , the point with the maximum weight in each of the $\sqrt{B}$ chunks. Thus a leaf stores $O(B)$ points and fits in $O(1)$ blocks. Similarly, an internal node of Ψ_v^1 stores for each of the $\sqrt{B}$ slabs the point with the maximal weight in each of the subtrees rooted in its $\sqrt{B}$ children. Therefore an internal node also fits in $O(1)$ blocks, and Ψ_v^1 uses $O(n_v / (\log_B n \sqrt{B})) = O(n_v / (\log_B n))$ blocks in total.

Consider a multi-slab $\sigma_v[i : j]$. To find the the maximum weight in chunks $C_a, \dots, C_b$ spanned by a rank range $[\rho_1, \rho_2]$, we visit the nodes on the paths from the root of Ψ_v^1 to the leaves corresponding to C_a and C_b . In each of these $O(\log_B n)$ nodes we consider the points contained in both multi-slab $\sigma_v[i : j]$ and one of the chunks $C_a, \dots, C_b$, and select the maximal weight point. This takes $O(1)$ I/Os. Finally, we select the maximum of the $O(\log_B n)$ weights.

This completes the description of our static two-dimensional range max structure. In the full version of the paper we describe how it can be constructed in $O(n \log_B n)$ I/Os in a bottom-up, level-by-level manner.

Theorem 1. *A set of N points in the plane can be stored in a linear-size structure such that an orthogonal range-max query can be answered in $O(\log_B^2 n)$ I/Os. The structure can be constructed in $O(n \log_B n)$ I/Os.*

Dynamization. Next we sketch how to make our data structure dynamic. Details will appear in the full paper.

To delete a point p from S we delete it from the relevant $O(\log_B n)$ $\mathcal{M}_v$ structures as well as from the base tree. The latter is done in $O(\log_B n)$ I/Os

using global rebuilding [17]. To delete p from a $\mathcal{M}_v$ structure we need to delete it from Ψ_v^1, Ψ_v^2 , and Ψ_v^3 . Since we cannot update a Cartesian tree efficiently, which is the building block of Ψ_v^2 , we modify the structure so that we no longer partition each chunk C_k of P_v into minichunks (that is, we remove Ψ_v^2). Instead we construct $\Psi_v^3[k]$ directly on the points in C_k . This allows us to delete p from $\mathcal{M}_v$ in $O(\log_B n)$ I/Os: We first delete p from Ψ_v^3 by marking its weight rank ω_p as ∞ , and then update Ψ_v^1 if necessary. However, since $|C_k| \leq B \log_B N$, $\Psi_v^3[k]$ now uses $O(\log_B \log_B n)$ blocks and the overall size of the structure becomes $O(n \log_B \log_B n)$ blocks. The construction cost becomes $O(n \log_B n \log_{M/B} \log_B n)$ I/Os.

To handle insertions we use the external logarithmic method [6]; This way an insertion takes $O(\log_B^2 n \log_{M/B} \log_B n)$ I/Os amortized and the query cost is increased by a factor of $O(\log_B n)$.

Theorem 2. *A set of N points in the plane can be stored in a structure that uses $O(n \log_B \log_B n)$ disk blocks such that a range-max query can be answered in $O(\log_B^3 n)$ I/Os. A point can be inserted or deleted in $O(\log_B^2 n \log_{M/B} \log_B n)$ and $O(\log_B^2 n)$ I/Os amortized, respectively.*

In the full paper we describe various extensions and improvements. For example, by using Cartesian trees to implement Ψ_v^1 and a technique to speed up the identification process [10], we can improve the query bound of our linear-size static structure to $O(\log_B^{1+\epsilon} n)$ I/Os. However, we cannot construct this structure efficiently and therefore cannot make it dynamic.

3 Stabbing-Max Queries

In Section 3.1 we describe our stabbing-max structure for the one-dimensional case, and in Section 3.2 we sketch how to extend it to two dimensions.

3.1 One-Dimensional Structure

Given a set S of N intervals, where each interval $\gamma \in S$ is assigned a weight $w(\gamma)$, we want to compute the maximum-weight interval in S containing a query point. Our structure for this problem is based on the external interval tree of Arge and Vitter [7], as well as on the ideas utilized in the point-location structure of Agarwal et al [1]. We are mainly interested in the dynamic case, since the static version of the problem is easily solved.

Overall structure. Our structure consists of a fanout $\sqrt{B}$ base B-tree T on the endpoints of the intervals in S , with the intervals stored in secondary structures associated with the internal nodes of T . Each leaf represents B consecutive points and the tree has height $O(\log_B n)$. As in Section 2, a canonical interval σ_v is associated with each node v ; σ_v is partitioned into $k \leq \sqrt{B}$ slabs by the ranges $\sigma_{v_1}, \dots, \sigma_{v_k}$ associated with the children $v_1, v_2, \dots, v_k$ of v . An input interval γ is assigned to v if $\gamma \subseteq \sigma_v$ but $\gamma \not\subseteq \sigma_{v_i}$ for any $1 \leq i \leq k$. A leaf z stores intervals

whose both endpoints lie in σ_z . The $O(B)$ intervals S_z assigned to z are stored using $O(1)$ blocks. At each internal node v , $\Theta(\sqrt{B})$ secondary structures are used to store the set of intervals S_v assigned to v . A *left-slab structure* $\mathcal{L}_v[i]$ and a *right-slab structure* $\mathcal{R}_v[i]$, for each of the $\sqrt{B}$ slabs, and a *multi-slab structure* $\mathcal{M}_v$. $\mathcal{L}_v[i]$ (resp. $\mathcal{R}_v[i]$) contains intervals from S_v whose left (resp. right) endpoints lie in σ_{v_i} . It supports stabbing queries for points in σ_{v_i} in $O(\log_B n)$ I/Os. The multi-slab structure $\mathcal{M}_v$ stores all intervals that span at least one slab. For any query point $q \in \sigma_{v_i}$, it can be used to find the maximum-weight interval that completely spans σ_{v_i} in $O(1)$ I/Os. We describe the slab and multi-slab structures below. Refer to Figure 3(a). Overall, an interval is stored in at most three secondary structures, and each secondary structure uses linear space, therefore the overall structure also uses linear space.

Answering a query. To report the maximum-weight interval containing a query point q , we search down the base tree T for the leaf z containing q . At each of the $O(\log_B n)$ nodes v on the path, we compute the maximum-weight interval of S_v containing q and return the maximum-weight interval of these $O(\log_B n)$ intervals. To answer a query at an internal node v with $q \in \sigma_{v_i}$, we simply query the left-slab structure $\mathcal{L}_v[i]$ and right-slab structure $\mathcal{R}_v[i]$ to compute the maximum-weight interval whose one endpoint lies in σ_{v_i} and that contains q . We then query the multi-slab structure $\mathcal{M}_v$ to compute the maximum-weight interval spanning σ_{v_i} . Refer to Figure 3(b). At the leaf z we simply scan the $O(B)$ intervals stored at z to find the maximum. Since we spend $O(\log_B n)$ I/Os in each node, we answer a query in a total of $O(\log_B^2 n)$ I/Os.

Left/right-slab structure. Let $R_v^i \subseteq S_v$ be the set of intervals whose right endpoints lie in σ_{v_i} . These intervals are stored in the right-slab structure $\mathcal{R}_v[i]$. Answering a stabbing query on R_v^i with a point $q \in \sigma_{v_i}$ is equivalent to answering a one-dimensional range max query $[q, \infty]$ on the right endpoints of R_v^i . Refer to Figure 3(c). As discussed in Section 2, such a query can easily be answered in $O(\log_B n)$ I/Os using a B-tree. $\mathcal{L}_v[i]$ is implemented in a similar way.

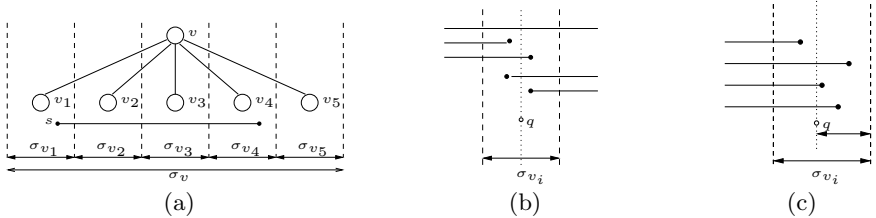


Fig. 3. (a) Node v in the base tree. The range σ_v associated with v is divided into 5 slabs. Interval s is stored in the left slab structure corresponding to σ_{v_1} and the right slab structure corresponding to σ_{v_4} , as well as in the multi-slab structure $\mathcal{M}_v$. (b) Querying a node with q . (c) Equivalence between a stabbing-max query q and a one-dimensional range max query $[q, \infty]$.

Multi-slab structure. A multi-slab structure $\mathcal{M}_v$ stores intervals S'_v from S_v that span at least one slab. $\mathcal{M}_v$ is a fan-out $\sqrt{B}$ B-tree on S'_v ordered by interval id's. For a node $u \in \mathcal{M}_v$, let γ_{ij} be the maximum-weight interval that spans σ_{v_i} and that is stored in the subtree rooted at the j -th child of u . For $1 \leq i, j \leq \sqrt{B}$, we store γ_{ij} at u . In particular, the root of $\mathcal{M}_v$ stores the maximum-weight interval spanning each of the $\sqrt{B}$ slabs, and a stabbing query in any slab σ_{v_i} can therefore be answered in $O(1)$ I/Os. Since each node can be stored in $O(1)$ blocks, $\mathcal{M}_v$ uses linear space. Note how $\mathcal{M}_v$ corresponds to “combining” $\sqrt{B}$ B-trees with fan-out $\sqrt{B}$ in a single B-tree.

To insert or delete an interval γ , we first search down $\mathcal{M}_v$ to find and update the relevant leaf z . After updating z , some of the intervals stored at nodes on the path P from the root of $\mathcal{M}_v$ to z may need to be updated. To maintain a balanced tree, we also perform B-tree rebalancing operations on the nodes on P . Both can easily be done in $O(\log_B n)$ I/Os in a traversal of P from z towards the root, as in [6].

Dynamization. To insert a new interval γ we first insert the endpoints of γ in T . By implementing T as a weight-balanced B-tree we can do so in $O(\log_B n)$ I/Os. Refer to [7] for details. Next, we use $O(\log_B n)$ I/Os to search down T for the node v where γ needs to be inserted in the secondary structures. Finally, we use another $O(\log_B n)$ I/Os to insert γ in a left and right slab structure, as well as in the multi-slab structure $\mathcal{M}_v$ if it spans at least one slab. To delete an interval γ we first delete it from the relevant secondary structures using $O(\log_B n)$ I/Os. Then we delete its endpoints from T using the global-rebuilding technique [17]. Since we can easily rebuild the structure in $O(n \log_B n)$ I/Os, this adds another $O(\log_B n)$ I/Os to the delete bound. Details will appear in the full paper.

Theorem 3. *A set of N intervals can be stored in a linear space data structure such that a stabbing-max query can be answered in $O(\log_B^2 n)$ I/Os, and such that updates can be performed in $O(\log_B n)$ I/Os. The structure can be constructed in $O(n \log_B n)$ I/Os.*

In the full paper we describe various extensions and improvements. For example, we can easily modify our structure to handle *semigroup stabbing queries*. Let $(\mathbf{S}, +)$ be a commutative semigroup. Given a set of N intervals S , where interval $\gamma \in S$ is assigned a weight $w(\gamma) \in \mathbf{S}$, the result of a semigroup stabbing query q is $\sum_{q \in \gamma, \gamma \in S} w(\gamma)$. Max queries is the special case where the semigroup is taken to be $(\mathbb{R}, \max)$. Unlike the structure presented in this section, the 2D range-max structure described in Section 2 cannot be generalized, since it utilizes that in the semigroup $(\mathbb{R}, \max)$ the result of a semigroup operation is one of the operands.

By combining the ideas used in our structure with ideas from the external segment tree of Arge and Vitter [7], we can also obtain a space-time tradeoff. More precisely, for any $\epsilon > 0$, a set of N intervals can be stored in a structure that uses $O(n \log_B^\epsilon n)$ disk blocks, such that a stabbing-max query can be answered in $O(\log_B^{2-\epsilon} n)$ I/Os and such that updates can be performed in $O(\log_B^{1+\epsilon} n)$ I/Os amortized.

3.2 Two-Dimensional Structure

In the two-dimensional stabbing-max problem we are given a set S of N weighted rectangles in $\mathbb{R}^2$, and want to be able to find the maximal-weight rectangle containing a query point q . We can extend our one-dimensional structure to this case using our one-dimensional stabbing-max and two-dimensional range-max structures. For space reasons we only give a rough sketch of the extension.

The structure consists of a base B-tree T with fanout $B^{1/3}$ on the x -coordinates of the corners of the rectangles in S . As in the 1D case, an interval σ_v is associated with each node v , and this interval is partitioned into $B^{1/3}$ vertical slabs by its children. A rectangle γ is stored at an internal node v of T if $\gamma \subseteq \sigma_v$ but $\gamma \not\subseteq \sigma_{v_i}$ for any child v_i of v . Each internal node v of T stores a multi-slab structure and one left- and right-slab structure for each slab. A multi-slab structure stores rectangles that span slabs and the left-slab (right-slab) structures of the i -th slab σ_{v_i} at v stores rectangles whose left (right) edges lie in σ_{v_i} .

The slab and multi-slab structures are basically one-dimensional stabbing-max structures on the y -projections of those rectangles. For the multi-slab structure we utilize the same “combining” technique as in the one-dimensional case to conceptually build a one-dimensional structure for each slab. The decreased fanout of $B^{1/3}$ allows us to use only linear space while being able to answer a query in $O(\log_B^2 n)$ I/Os. For the slab structures we utilize our two-dimensional range-max structure to be able to answer a query in $O(\log_B^3 n)$ I/Os. Details will appear in the full paper.

We answer a stabbing-max query by visiting $O(\log_B n)$ nodes on a path in T , and querying two slab structures and the multi-slab structure in each node. Overall, a query is answered in $O(\log_B^4 n)$ I/O. As previously, we can also make the structure dynamic using the external logarithmic method. Again details will appear in the full paper.

Theorem 4. *A set of N rectangles in $\mathbb{R}^2$ can be stored in a linear-size structure such that stabbing-max queries can be answered in $O(\log_B^4 n)$ I/Os.*

A set of N rectangles in $\mathbb{R}^2$ can be stored in a structure using $O(n \log_B \log_B n)$ disk blocks such that stabbing-max queries can be answered in $O(\log_B^5 n)$ I/Os, and such that insertions and deletions can be performed in $O(\log_B^2 n \log_{M/B} \log_B n)$ and $O(\log_B^2 n)$ I/Os amortized, respectively.

References

1. P. K. Agarwal, L. Arge, G. S. Brodal, and J. S. Vitter. I/O-efficient dynamic point location in monotone planar subdivisions. In *Proc. ACM-SIAM Symp. on Discrete Algorithms*, pages 1116–1127, 1999.
2. P. K. Agarwal, L. Arge, and S. Govindarajan. CRB-tree: An optimal indexing scheme for 2D aggregate queries. In *Proc. Intl. Conf. on Database Theory*, 2003.
3. P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry* (B. Chazelle, J. Goodman, R. Pollack, eds.), pages 1–56. American Mathematical Society, Providence, RI, 1999.

4. A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Comm. ACM*, 31(9):1116–1127, 1988.
5. L. Arge. External memory data structures. In *Handbook of Massive Data Sets*, pages 313–358. Kluwer Academic Publishers, 2002.
6. L. Arge and J. Vahrenhold. I/O-efficient dynamic planar point location. In *Proc. ACM Symp. on Computational Geometry*, pages 191–200, 2000.
7. L. Arge and J. S. Vitter. Optimal dynamic interval management in external memory. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 560–569, 1996.
8. R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.
9. J. L. Bentley. Multidimensional divide and conquer. *Comm. ACM*, 23(6):214–229, 1980.
10. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, June 1988.
11. D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.
12. H. Edelsbrunner and H. A. Maurer. On the intersection of orthogonal objects. *Information Processing Letters*, 13:177–181, 1981.
13. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
14. R. Grossi and G. F. Italiano. Efficient cross-tree for external memory. In *External Memory Algorithms and Visualization*, pp. 87–106. AMS, DIMACS series in Discrete Mathematics and Theoretical Computer Science, 1999.
15. K. V. R. Kanth and A. K. Singh. Optimal dynamic range searching in non-replicating index structures. In *Proc. Intl. Conf. on Database Theory, LNCS 1540*, pages 257–276, 1999.
16. H. Kaplan, E. Molad, and R. E. Tarjan. Dynamic rectangular intersection with priorities. In *Proc. ACM Symp. on Theory of Computation*, pages 639–648, 2003.
17. M. H. Overmars. *The Design of Dynamic Data Structures*. Springer-Verlag, LNCS 156, 1983.
18. J. Robinson. The K-D-B tree: A search structure for large multidimensional dynamic indexes. In *Proc. SIGMOD Intl. Conf. on Management of Data*, pages 10–18, 1981.
19. J. Vuillemin. A unifying look at data structures. *Comm. ACM*, 23:229–239, 1980.
20. J. Yang and J. Widom. Incremental computation and maintenance of temporal aggregates. In *Proc. IEEE Intl. Conf. on Data Engineering*, pages 51–60, 2001.

Line System Design and a Generalized Coloring Problem

Mansoor Alicherry and Randeep Bhatia

Bell Labs, Lucent Technologies, Murray Hill, NJ 07974.
{mansoor,randeep}@research.bell-labs.com

Abstract. We study a generalized coloring and routing problem for interval and circular graphs that is motivated by design of optical line systems. In this problem we are interested in finding a coloring and routing of “demands” of minimum total cost where the total cost is obtained by accumulating the cost incurred at certain “links” in the graph. The colors are partitioned in sets and the sets themselves are ordered so that colors in higher sets cost more. The cost of a “link” in a coloring is equal to the cost of the most expensive set such that a demand going through the link is colored with a color in this set. We study different versions of the problem and characterize their complexity by presenting tight upper and lower bounds. For the interval graph we show that the most general problem is hard to approximate to within $\sqrt{s}$ and we complement this result with a $O(\sqrt{s})$ -approximation algorithm for the problem. Here s is proportional to the number of color sets. For the circular graph problem we show that most versions of the problem are hard to approximate to any bounded ratio and we present a $2(1 + \epsilon)$ approximation scheme for a special version of the problem.

1 Introduction

The basic graph coloring problem, where the goal is to minimize the number of colors used has been extensively studied in the literature. Interval graph coloring and Circular graph coloring [5] are two special cases where the former can be solved in polynomial time and the latter is known to be NP-Hard [4], [5]. Recently some generalizations of the basic graph coloring problem have received much attention in the literature. In the minimum sum coloring (MSC) [10], [12] problem we are interested in coloring the graph with natural numbers such that the total sum of the colors (numbers) assigned to the vertices is minimized. A generalization of the MSC problem is the Optimum Cost Chromatic Partition (OCCP) [21] problem where we are interested in coloring the graph so as to minimize the total cost of the colors assigned to all the vertices where the i -th color has cost k_i . These problems have been shown to be NP-hard [12], [20] and even quite hard to approximate [11], [1], [7] for general graphs. However polynomial time algorithms are known for trees [12], [9]. These problems have also been studied for interval graphs [15], [6], [9], [7] and bipartite graphs [2], [7].

In this paper we study a generalized coloring problem, for interval and circular arc graphs, which is motivated by an optical line system (OLS) design

problem. In this problem we are interested in finding a coloring (wavelength assignment) and routing (for circular graphs) of intervals (demands) of minimum total cost where the total cost is obtained by accumulating the cost incurred at certain points (amplifier locations) on the number line (circle). The colors are partitioned in sets and the sets themselves are ordered (by their associated cost) so that colors in higher sets cost more. The cost incurred at a point p is equal to the cost of the highest set s , such that there exists an interval i containing point p , where i is colored with one of the colors in s . We study different versions of the problem and characterize their complexity by presenting tight upper and lower bounds.

Optical Line Systems (OLS) allow for transporting large amounts of data over long spans of optical fiber, by multiplexing and demultiplexing optical wavelengths using what are called *end terminals* (ET). Wavelengths are selectively added or dropped at intermediate points, using devices called *optical add-drop multiplexers* (OADM). (see figure 1). Demands for an OLS originate (or terminate) at the ETs or at the OADMs. Each demand requires a wavelength and multiple demands sharing a fiber span must be allocated different wavelengths. To prevent degradation in signal quality, *Optical amplifiers*, capable of amplifying the signal, are placed at the intermediate points between the end terminals. The type and cost of the amplifier required at a location, depends on the wavelengths assigned to the demands routed via the location. Specifically the cost is more the higher the wavelength to be amplified and each amplifier type is capable of amplifying all the wavelengths upto a certain maximum. The OLS design problem is to find a valid wavelength assignment and routing for the demands so that the total cost of the amplifiers needed is minimized. Related to the OLS design problem for rings is the problem of designing minimum cost optical ring system (SONET rings) [3], [19], [8], [14] with OADMs, to satisfy pairwise traffic demands. The problem of wavelength assignment in optical line systems in the context of fiber minimization is studied in [22]. The problem of routing and wavelength assignment in optical networks is extensively studied [18], [16].

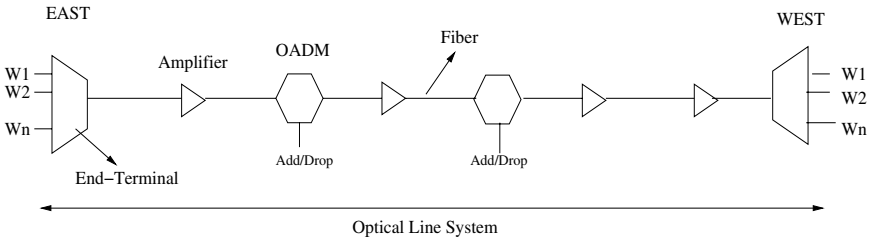


Fig. 1. A schematic diagram of an optical line system

2 Problem Description and Our Results

We formulate two class of problems depending on if the underlying line system is linear or circular. Here we present the problem mainly for the linear line system and point out the differences for the circular line system. A *Linear (Circular) Line System* consisting of n amplifiers is modeled by a number line (*circle*) labeled with $n + 1$ (n) points or nodes, such that there is an amplifier between any two adjacent nodes. The amplifiers are modeled by links between adjacent nodes. Thus the i -th link, denoted by e_i , corresponds to the i -th amplifier and connects the $i - 1$ -th and i -th ($i \bmod n$) node from the left (*in a clockwise traversal of nodes which are ordered in a clockwise traversal*).

The set of demands that are to be supported by the line system is denoted by D , where each demand in D is between two nodes of the system. For a Linear (*Circular*) Line System a demand is thus an interval $[x_1, x_2]$, $x_1 < x_2$ such that x_1 and x_2 are the coordinates of two nodes on the number line (*circle*). We also represent a demand by a tuple (i, j) , $i \leq j$ where for a Linear (*Circular*) Line System the demand (i, j) must be routed through links $e_i, e_{i+1}, \dots, e_j$ (*routed either clockwise on links $e_i, e_{i+1}, \dots, e_j$ or anti-clockwise on links $e_{j+1}, e_{j+2}, \dots, e_n, e_1, e_2, \dots, e_{i-1}$*).

The line system is assumed to have r wavelengths (colors) $1, 2, \dots, r$. These r colors are partitioned into k sets $C_1, C_2, \dots, C_k$, whose colors form a non-decreasing sequence. Thus for all $i < j$ we have $a < b$ for all $a \in C_i$ and $b \in C_j$. The cost of set C_i is i . If $h_j \in C_i$ is the largest wavelength (color) assigned to some demand routed over link e_j then the cost for link e_j is $c(e_j) = i$. The total cost of the line system is then $\sum_j c(e_j)$. The Linear (*Circular*) Line System Design Problem LLSDP (*CLS DP*) is to color and route demands such that no two demands routed on the same link get the same color and the total cost of the line system is minimized.

We define the load l_i^R of a link e_i for a given routing R of the demands D to be the number of demands using link e_i in R . We define the load l^R of the line system for routing R as $l^R = \max l_i^R$. Note that there is a unique routing R for the LLSDP, and we denote by $l = l^R$ the load of the line system, and by $l_i = l_i^R$ the load of link e_i . For CLSDP let R be the routing for which l^R is minimized. Then the load of the line system for the CLSDP is denoted by $l = l^R$. Let $l \in C_s$, then for any routing, there is a link with cost at least s . We call s the step requirement for the problem. We assume that the number of different color sets k is at most $c.s$ for some constant c .

We use the notation (α, β, γ) to denote the different problems that we consider in this paper. $\alpha = L$ or $\alpha = C$ depending on whether the underlying problem is LLSDP or CLSDP respectively. $\beta = U$ or $\beta = D$ depending on whether all step sizes (cardinality of sets C_i) are the same or different respectively. $\gamma = E$ or $\gamma = NE$ depending on whether we can exceed the line system step requirement or not ($k > s$ or $k = s$) respectively. In other words in the latter case only colors in $\bigcup_{i=1}^s C_i$ are available for coloring the demands, while in the former case all colors are available for coloring the demands. We use the

Table 1. Bounds for line system design problems

General Case			Special Case	
problem	Approx lower bound	Approx upper bound	problem	Complexity
$(L, D, *)$	$\Omega(\sqrt{s})$	$O(\sqrt{s})$	$(L, *, E), s = 2$	polynomial
(L, U, NE)	$1 + 1/s^2$	2	$ C_2 = \infty$	
(L, U, E)	NP-hard	2	$(L, *, NE), s = 2$	polynomial
$(C, *, NE)$	in-approximable		$(L, U, E), s = 2$	4/3-Approx
(C, D, E)	in-approximable		$(L, D, *), s = 3$	NP-hard
(C, U, E)	NP-hard	$2(1 + \epsilon)$		

wild-card $*$ to indicate all possible values. Our results for different version of the problem are summarized in Table 1.

3 Algorithms

In this section we present efficient optimal and approximation algorithms for the different versions of the Line System Design Problem. We say that in a coloring of the demands a link e_i is colored with t steps if all of the demands through link e_i are colored with colors in $\bigcup_{j=1}^t C_j$ and some demand through link e_i is colored with a color in C_t . Note that we can assume without loss of generality that $l_i > 0$ for all e_i . In this section we represent a demand by a tuple (i, j) , $i \leq j$ where for a Linear Line System the demand (i, j) must be routed through links $e_i, e_{i+1}, \dots, e_j$.

3.1 2-Approximation for the $(L, U, *)$ Problems

We present an algorithm A for these problems. The algorithm A works in phases where in each phase A colors some demands with at most two new colors, assigned in the order $1, 2, \dots, r$. The colored demands are removed from the line system to prepare for the next phase. Let $l(p)$ and $l_i(p)$ denote the load of the line system and the load of link e_i respectively at the beginning of phase p . Note that $l(1) = l$ and $l_i(1) = l_i$, for all i . We assume that at the beginning of each phase $l_i(p) \geq 1$ for all links e_i for the given instance of the LLSDP. This is because if some $l_i(p) = 0$ then the LLSDP instance can be sub-divided into two LLSDP instances, one for links $e_1, e_2, \dots, e_{i-1}$ and one for links $e_{i+1}, e_{i+2}, \dots, e_n$, which can be independently solved. In phase p for $l(p) \geq 2$ algorithm A constructs a directed multi-graph $G = (V, E)$ of n nodes with unit edge capacities in which 2 unit of flow can be routed from a source to a sink node. The nodes $V = \{0, 1, \dots, n-1\}$. For every demand $(i, j) \in D$ that is still uncolored in this phase a directed edge $(i-1, j)$ of unit capacity is added to E . For every link e_i for which $l_i(p) < l(p)$ an edge $(i-1, i)$ of unit capacity is in E . Node 0 is the source node and node $n-1$ is the sink node. It is easy to see that 2-units of

flow can be routed in the graph since every cut between the source and sink has capacity at least 2 and moreover since all edge capacities are integral this flow is routed over exactly two paths P_1 and P_2 . Let the smallest index of the color not used by A in phase p be $m(p)$ ($m(p) = 1$ in the first phase). In phase p , A assigns color $c_{m(p)}$ to all demands for which there is an edge in P_1 and assigns color $c_{m(p)+1}$ to all demands for which there is an edge in P_2 . For the next phase we have $m(p+1) = m(p) + 2$. Let d_i be the number of demands through edge e_i that are assigned color in phase p . Note that $d_i \leq 2$. Then $l_i(p+1) = l_i(p) - d_i$ for edge e_i and $l(p+1)$ is set to the maximum $l_i(p+1)$. In the case where $l(p) = 1$ then in phase p of the algorithm all the uncolored demands are non-overlapping and A colors them with the smallest available color.

Theorem 1. *Algorithm A is a 2-approximation for the $(L, U, *)$ problems.*

Proof. Note that $l(p+1) = l(p) - 2$ for all phases p for which $l(p) \geq 2$. This is because for every link e_i for which $l_i(p) = l(p)$ we have $l_i(p+1) = l_i(p) - 2$ and for every link e_i for which $l_i(p) < l(p)$ we have $l_i(p+1) \leq l_i(p) - 1$. Also note that $l(p) = 1$ implies $l(p+1) = 0$. Thus all the demands are colored using l colors implying that the coloring is feasible for both the (L, U, E) and (L, U, NE) problems. We show that all demands that go through edge e_i are colored in the first l_i phases. Note that at phase p , $l_i(p)$ is equal to l_i minus the number of demands through link e_i that have been colored in first $p-1$ phases. Also for all the links e_i for which $l_i(p) > 0$ at least one demand going through link e_i is colored in phase p . Thus $l_i(p) = 0$ at some phase $p \leq l_i$. Hence all demands that pass through edge e_i are colored by phase l_i . This implies that the largest index of the colors assigned to demands going through link e_i by A is at most $2l_i$. Hence the cost of link e_i in this coloring is at most $c(2l_i)$. Note that in any coloring of the demands the cost of link l_i is at least $c(l_i)$. Since all k color sets $C_1, C_2, \dots, C_k$ have the same cardinality for the uniform step problem we have $c(2l_i) \leq 2c(l_i)$. This implies that the cost of the line system as obtained by algorithm A is at most twice the cost of the optimal line system.

3.2 $2(1 + \epsilon)$ -Approximation for the (C, U, E) Problem for Constant Step Size

Note that this problem has two aspects: one of selecting a routing for each demand (clockwise or anti-clockwise) and one of coloring the routed demands. The algorithm for solving this problem decouples these two aspects and works in two phases. In the first phase the algorithm computes a routing R of the demands and in the second phase colors the routed demands. We describe these two phases separately. In the following we let $L(R) = \sum_{i=1}^n c(l_i^R)$ denote the load based lower bound on the cost of routing R .

The routing phase: Let $\epsilon > 0$ be given. Let S denote the size of each step ($|C_i| = S, \forall i$). We assume S is a constant. Let the shortest path routing R_s be defined as a routing in which every demand is routed in the direction in which it goes through smaller number of links (ties broken arbitrarily). If the

cost lower bound $L(R_s) \geq n(1 + \epsilon)/\epsilon$, then R_s is the routing output by the algorithm. Otherwise the set of demands D is partitioned into two sets D_1 and D_2 . Here $d \in D_1$ if and only if d goes through at least $n/3$ links in any of the two possible routings of d . The algorithm tries all possible routings $\mathcal{R}$ in which demands in D_1 are routed in either direction while at most $3S$ demands in the set D_2 are routed in the direction where they go through more links (not on the shortest path). Let $R \in \mathcal{R}$ be a routing for which the cost lower bound $L(R)$ is minimized. The algorithm outputs routing R . Let R^* be a routing for which $L(R^*) = \min_R L(R)$.

Claim. If the cost lower bound $L(R_s) \geq n(1 + \epsilon)/\epsilon$, then $L(R_s) \leq (1 + \epsilon)L(R^*)$.

Proof. Note that by definition of R_s we have $\sum_{i=1}^n l_i^{R_s} \leq \sum_{i=1}^n l_i^{R^*}$. Also note that $c(l_i^R) = \lceil \frac{l_i^R}{S} \rceil$. Thus $\sum_{i=1}^n c(l_i^{R_s}) \leq \sum_{i=1}^n c(l_i^{R^*}) + n$. Or $L(R_s) \leq L(R^*) + n$. Since $n(1 + \epsilon)/\epsilon \leq L(R_s)$ we have $n/\epsilon \leq L(R^*)$. Thus $n \leq L(R^*)\epsilon$. Hence $L(R_s) \leq L(R^*) + L(R^*)\epsilon = (1 + \epsilon)L(R^*)$.

Claim. If the cost lower bound $L(R_s) \leq n(1 + \epsilon)/\epsilon$, then $|D_1| \leq 3S(1 + \epsilon)/\epsilon$.

Proof. Note that $\sum_{i=1}^n l_i^{R_s} \leq SL(R_s) \leq Sn(1 + \epsilon)/\epsilon$. Also note that each demand in D_1 must go through at least $n/3$ links in any routing and in particular in R_s . Hence $|D_1|n/3 \leq \sum_{i=1}^n l_i^{R_s}$. Thus $|D_1|n/3 \leq Sn(1 + \epsilon)/\epsilon$ implying the claimed bound.

Claim. If the cost lower bound $L(R_s) \leq n(1 + \epsilon)/\epsilon$, then in R^* at most $3S$ demands in D_2 are routed in the longer direction (where they go through more links).

Proof. Note that by definition of R_s we have $\sum_{i=1}^n l_i^{R_s} \leq \sum_{i=1}^n l_i^{R^*}$. Let $D_3 \subseteq D_2$ be the set of demands in D_2 that are routed in the longer direction in R^* . Thus $\sum_{i=1}^n l_i^{R_s} + |D_3|n/3 \leq \sum_{i=1}^n l_i^{R^*}$ since each demand in D_3 goes through $2n/3 - n/3 = n/3$ more links on the longer path than the shorter path. Hence $\sum_{i=1}^n l_i^{R_s}/S + |D_3|n/3S \leq \sum_{i=1}^n \lceil l_i^{R^*}/S \rceil = L(R^*)$. However $L(R_s) = \sum_{i=1}^n \lceil l_i^{R_s}/S \rceil \leq \sum_{i=1}^n l_i^{R_s}/S + n$. Thus $L(R_s) - (n - |D_3|n/3S) \leq L(R^*)$. Since $L(R^*) \leq L(R_s)$ we must have $0 \leq n - |D_3|n/3S$ or $|D_3| \leq 3S$.

Corollary 1. *The routing R output by the algorithm satisfies $L(R) \leq (1 + \epsilon)L(R^*)$.*

Proof. If the cost lower bound $L(R_s) \leq n(1 + \epsilon)/\epsilon$, then the algorithm outputs routing R^* , for which $L(R^*) = \min_R L(R)$. Otherwise by Claim 3.2 the routing R_s output by the algorithm satisfies $L(R_s) \leq (1 + \epsilon)L(R^*)$.

Corollary 2. *The running time of the routing phase of the algorithm is $O(2^{3P}(nP + 3S)^{3S})$ where $P = S(1 + \epsilon)/\epsilon$.*

Proof. The proof follows from the observation that $L(R_s) \leq n(1 + \epsilon)/\epsilon$ and that there are at most 2^{3P} ways of routing demands in D_1 .

Coloring Phase: Let R be the routing output by the routing phase of the algorithm. The coloring phase of the algorithm itself is sub-divided into at most two phases.

An iteration of the first phase is invoked as long the uncolored demands in R go through all links, and involves repeatedly finding a subset of uncolored demands $d_0, d_1, \dots$ that can be colored with 2 colors and that go through all the n links in R . These demands are then colored with the smallest available two colors (that have not been used for coloring any other demands). Demand d_0 is an uncolored demand that covers the largest number of links in R . Demand d_1 is an uncolored demand that overlaps with d_0 and covers the largest number of uncovered links in R , in the clockwise direction. Demand d_2 overlaps with d_1 in R and covers the largest number of uncovered links in R and so on until all links are covered by the selected demands. It is easy to see that in each iteration demands $d_0, d_1, \dots$ are 2-colorable.

Let e_j be an uncovered link in the beginning of the second sub-phase. It is easy to see that if link e_j is removed we get an instance of the (L, U, E) problem, for the uncolored demands, which is solved using the algorithm A presented in Section 3.1.

Claim. Let R be the routing output by the routing phase of the algorithm. Then the cost of the coloring output by the coloring phase of the algorithm is at most $2L(R)$.

Proof. The proof is along the line of the proof for Theorem 1 for algorithm A in Section 3.1 and is based on the observation that all the demands through a link e_i with load l_i^R are colored with the first $2l_i^R$ colors and since $c(2l_i^R) \leq 2c(l_i^R)$ which follows from the fact that all step sizes are the same.

Theorem 2. *The presented algorithm is a $2(1 + \epsilon)$ -approximation for the (C, U, E) problem with constant step size.*

Proof. Let O be the optimal solution and let R_O be the routing used by O . Let R be the routing used by the solution output by the algorithm. Then by Corollary 1 we have $L(R) \leq (1 + \epsilon)L(R_O)$. Note that the cost of the optimal solution O is at least $L(R_O)$. By Claim 3.2 the cost of the solution output by the algorithm is at most $2L(R)$. Combining these together we get the claimed bound.

3.3 Other Algorithms (Proofs Omitted)

Claim. For $k = 2$, $(L, *, E)$ problems with $|C_2| = \infty$ and $(L, *, NE)$ problems are optimally solvable in polynomial time.

The algorithm uses a combination of the flow technique given in section 3.1 and dynamic programming.

Claim. There exist a $O(\sqrt{s})$ -approximation algorithm for $(L, *, *)$ problems.

The algorithm works by creating an instance of problem (L, D, NE) with two steps, which is solved optimally (Claim 3.3), and its solution when mapped back to the original problem, is shown to be within a factor $O(\sqrt{s})$ of the optimal.

Claim. There exists a $4/3$ -approximation for the (L, U, E) problem when $s = 2$ for $k \geq 3$

The algorithm works by selecting the best of two solutions O_1 and O_2 , where O_1 is obtained by coloring the demands, with colors in $C_1 \cup C_2$. O_2 is obtained by solving a $(L, *, E)$, $k = 2$ problem so as to maximize the number of links colored with one step only. The remaining uncolored demands are then colored with colors in $C_2 \cup C_3$.

Claim. For $(L, *, E)$ problems, there exists an optimal solution which does not use more than $2l$ colors.

4 Inapproximability Results for Linear Line System Design Problem

Design problems. In the following we denote a demand by an interval $[x_1, x_2]$, $x_1 < x_2$. We use the statement “insert a nodes in the interval $[i, j]$ ” to add $a + 1$ links to the line system between the points (or node) i and j . We use the standard notations (i, j) and $[i, j]$ respectively for open and closed intervals between point i and j . Motivated by the reduction in [9], we use reduction from the NP-complete [4] problem Numerical Three Dimensional Matching (N3DM) which is defined as follows.

N3DM. Given a positive integer t and $3t$ rational numbers a_i , b_i and c_i satisfying $\sum_{i=1}^t (a_i + b_i + c_i) = t$ and $0 < a_i, b_i, c_i < 1$ for $i = 1, \dots, t$, do there exist permutations ρ and σ of $\{1, \dots, t\}$ such that $a_i + b_{\rho(i)} + c_{\sigma(i)} = 1$ for $i = 1, \dots, t$?

4.1 (L, D, NE) for $s = 3$

Theorem 3. (L, D, NE) is NP-hard for $s = 3$.

Proof. The proof is illustrated with an example in figure 2, where the instance of N3DM is $(a_1, a_2) = (1/2, 1/3)$, $(b_1, b_2) = (1/3, 1/4)$ and $(c_1, c_2) = (1/4, 1/3)$. This instance has a solution $a_1 + b_2 + c_1 = a_2 + b_1 + c_2 = 1$. Let I_1 be an instance of N3DM containing the integer t and the rational numbers a_i, b_i and c_i for $i = 1, \dots, t$. Let A_i, B_j and $X_{i,j}$ be distinct rational numbers for $i, j = 1, \dots, t$ such that $3 < A_i < 4 < B_j < 5$ and $6 < X_{i,j} < 7$. For I_1 , an instance I_2 of LLSDP, with the underlying number line ranging from 0 to 13, is constructed as follows. The demands are:

1 of each $[0, A_i]$ for $i = 1, \dots, t$	1 of each $[B_j, X_{i,j}]$ for $i, j = 1, \dots, t$
$t - 1$ of each $[2, A_i]$ for $i = 1, \dots, t$	1 of each $[X_{i,j}, 8 + a_i + b_j]$ for $i, j = 1, \dots, t$
$t - 1$ of each $[1, B_j]$ for $j = 1, \dots, t$	1 of each $[9 - c_k, 13]$ for $k = 1, \dots, t$
1 of each $[2, B_j]$ for $j = 1, \dots, t$	$t^2 - t$ of each $[10, 12]$
1 of each $[A_i, X_{i,j}]$ for $i, j = 1, \dots, t$	1 of each $[X_{i,j}, 11]$ for $i, j = 1, \dots, t$

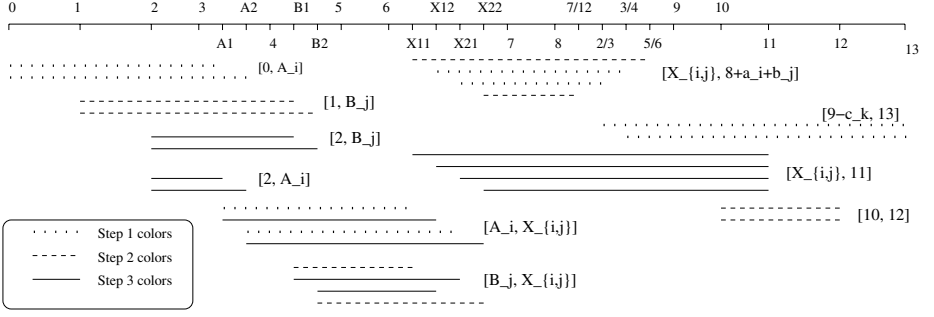


Fig. 2. An instance of LLSDP

In I_2 there are t colors in C_1 , $t^2 - t$ colors in C_2 and t^2 colors in C_3 . A node is placed at every point in the interval $[0, 13]$ wherever a demand starts or ends. Thus there is one node at each of the points $0, 1, 2, 10, 11, 12, 13$, t nodes in each of the intervals $(3, 4)$ and $(4, 5)$, t^2 nodes in the interval $(6, 7)$, and at most $t^2 + t$ nodes in the interval $(8, 9)$. The total number of nodes in the interval $[2, 11]$ is at most $2t^2 + 3t + 3$ and hence the total number of links in that interval is $2t^2 + 3t + 2$. We add $3(2t^2 + 3t + 1)$ additional nodes in each of the intervals $(1, 2)$ and $(11, 12)$ and add $6(2t^2 + 3t + 1)$ additional nodes in each of the intervals $(0, 1)$ and $(12, 13)$. It is easy to see that this reduction can be done in time polynomial in t .

We show that there is a solution for I_2 with cost $27(2t^2 + 3t + 2)$ or less if and only if the instance I_1 of N3DM has a solution. The load on each of the links in $[0, 1]$ and $[12, 13]$ is $t = |C_1|$ and there are $12(2t^2 + 3t + 2)$ links in these intervals. So the cost of the line system due to the links in these intervals is at least $12(2t^2 + 3t + 2)$. The load on each of the links in $[1, 2]$ and $[11, 12]$ is $t^2 = |C_1| + |C_2|$ and there are $6(2t^2 + 3t + 2)$ links in these intervals. So the cost of the line system due to links in these intervals has to be at least $12(2t^2 + 3t + 2)$. Hence it is easy to see that if the total cost of the line system is $27(2t^2 + 3t + 2)$ or less, then any demand that overlaps with the intervals $[0, 1]$ or $[12, 13]$ has to get colors only from C_1 and any demand that overlaps with the intervals $[1, 2]$ or $[11, 12]$ has to get colors only from $C_1 \cup C_2$. Thus the demands $[0, A_i]$ for $i = 1, \dots, t$ and $[9 - c_k, 13]$ for $k = 1, \dots, t$ must get colors from C_1 . Since each of the demands $[1, B_j]$ for $j = 1, \dots, t$ overlaps with each of the demands $[0, A_i]$ for $i = 1, \dots, t$ the demands $[1, B_j]$ for $j = 1, \dots, t$ must get colors from C_2 . Since there are at most $2t^2 + 3t + 2$ links in the interval $[2, 11]$ and there are only 3 color classes, the cost contribution of the links in this interval is at most $3(2t^2 + 3t + 2)$. Each of the the links in the interval $[2, 8]$ has a load of $2t^2 = |C_1| + |C_2| + |C_3|$ which is the maximum number of colors available on the system. Hence if a demand ends at a node in the interval $[2, 8]$, then the next demand that is colored with the same color starts from the same node (i.e. the demand fits seamlessly with its predecessor).

Assume that there is a solution for I_2 with cost $27(2t^2 + 3t + 2)$ or less. As we have shown, the demands $[0, A_i]$ for $i = 1, \dots, t$, and $[9 - c_k, 13]$ for $k = 1, \dots, t$ will get colors from C_1 . Assume that the demand $[0, A_i]$ gets color i . Let $[9 - c_{\sigma(i)}, 13]$ be the demand that gets color i among $[9 - c_k, 13]$ for $k = 1, \dots, t$. Note that σ forms a permutation of $\{1, \dots, t\}$. Note that the other demands that will get color i are $[A_i, X_{i,j}]$ and $[X_{i,j}, 8 + a_i + b_j]$ for some j , since these are the only demands that fit seamlessly with $[0, A_i]$. Call such a j as $\rho(i)$. We claim the following.

1. ρ is a permutation of $\{1, \dots, t\}$.
2. The demands $[X_{i,\rho(i)}, 8 + a_i + b_{\rho(i)}]$ and $[9 - c_{\sigma(i)}, 13]$ fit seamlessly.

For Claim 1, we want to show that $\rho(i_1) \neq \rho(i_2)$ for $i_1 \neq i_2$, thus implying that ρ forms a permutation. For contradiction let $\rho(i_1) = \rho(i_2) = j$ for $i_1 \neq i_2$. In this case both the demands $[X_{i_1,j}, 8 + a_{i_1} + b_j]$ and $[X_{i_2,j}, 8 + a_{i_2} + b_j]$ must get color from C_1 . Note that there are $t - 1$ copies of the demand $[1, B_j]$ which get color from C_2 and as shown before, the demands with end points in the interval $[2, 8]$, that are assigned the same colors, have to fit seamlessly. Thus for every one of the $t - 1$ copies of demand $[1, B_j]$, there exists a unique i in $\{1, \dots, t\}$ such that the color in C_2 assigned to the copy of the demand $[1, B_j]$ is the same as the color assigned to the demands $[B_j, X_{i,j}]$ and $[X_{i,j}, 8 + a_i + b_j]$. Thus, since there are t demands $[X_{i,j}, 8 + a_i + b_j]$ one for each value of i and since $t - 1$ of these must get color from C_2 it can't be the case that both the demands $[X_{i_1,j}, 8 + a_{i_1} + b_j]$ and $[X_{i_2,j}, 8 + a_{i_2} + b_j]$ get color from C_1 .

For Claim 2, note that if the demands $[X_{i,\rho(i)}, 8 + a_i + b_{\rho(i)}]$ and $[9 - c_{\sigma(i)}, 13]$ fit seamlessly, then $a_i + b_{\rho(i)} + c_{\sigma(i)} = 1$. If one of these pair of demands do not fit seamlessly for some $i = i_1$, then $a_{i_1} + b_{\rho(i_1)} + c_{\sigma(i_1)} < 1$, since the demands in a pair are assigned the same color and hence must not overlap. In this case, since $\sum_{i=1}^t (a_i + b_i + c_i) = t$, there exist another $i = i_2$ such that the demands $[X_{i_2,\rho(i_2)}, 8 + a_{i_2} + b_{\rho(i_2)}]$ and $[9 - c_{\sigma(i_2)}, 13]$ overlap, which contradicts the fact that these demands are assigned the same color. Hence from a solution of I_2 of cost $27(2t^2 + 3t + 2)$ or less, we can construct the solution to the instance I_1 of N3DM by looking at the demands having color from C_1 .

Conversely, given a feasible solution of I_1 , the construction can be reversed to find a solution of I_2 with cost $27(2t^2 + 3t + 2)$ or less. As N3DM is NP-complete, the interval graph coloring problem with three color classes is NP-hard.

We omit the proof of the following claims.

Claim. (L, D, NE) with $s = 3$ is in-approximable within a factor of $1 + \frac{1}{6}$ unless $P=NP$.

Claim. (L, D, E) with $s = 3$ is NP-hard.

4.2 Inapproximability of $(L, D, *)$ Problems (Proofs Omitted).

Theorem 4. (L, D, NE) is $\Omega(\sqrt{s})$ in-approximable.

The main idea is to create a line system problem with m overlapping copies of the instance I_2 created in the reduction given in Theorem 3 and with $s = m + \sqrt{m} + 1$ color classes. The copies are created in such a way that no two demands from different copies get the same color. The color classes are formed in such a way that for the links whose covering demands were colored with colors from three classes (C_1) , (C_2) , and (C_3) in the reduction in Theorem 3, their covering demands are colored from three classes (C_1) , $(C_2 \cup \dots \cup C_{\sqrt{m}+1})$ and $(C_{\sqrt{m}+2} \cup \dots \cup C_{\sqrt{m}+m+1})$ respectively, in this reduction. The intermediate links are placed such that we get an inapproximability ratio of $\Omega(\sqrt{m})$ and hence $\Omega(\sqrt{s})$.

In a similar way we can prove the following.

Theorem 5. (L, D, E) is $\Omega(\sqrt{s})$ in-approximable.

Theorem 6. $(L, U, *)$ is NP-hard and (L, U, NE) is $1 + 1/s^2$ in-approximable.

The proof uses a reduction from circular arc graph coloring problem.

5 Inapproximability of Circular Line System Design Problems (Proofs Omitted)

Claim. $(C, *, NE)$ is hard to approximate to any bounded ratio.

The main idea is the following. Let I_1 be an instance of circular arc graph coloring problem where the load is l everywhere along the circle. We create an instance I_2 of $(C, *, NE)$ with $r = l$ colors, such that I_2 has a solution if and only if I_1 is l -colorable. From I_1 we first create a new instance I'_1 of the circular arc graph coloring problem by cutting each arc into multiple *arc collections*, each of length less than a half-circle, such that the points at which the arcs are cut are distinct and they are not the end points of any of the existing arcs. The demands for I_2 are created, one for each arc of I'_1 , such that the end points of the demands are the end points of the arcs. Links are placed uniformly on the line system. Any solution to I_2 must route all demands in the direction of the arc and thus yield a l -coloring for I'_1 and hence for I_1 .

Claim. (C, D, E) is hard to approximate to any bounded ratio.

Claim. $(C, *, E)$ has same inapproximability ratio as $(L, *, E)$.

References

1. A. Bar-Noy, M. Bellare, M. M. Halldorsson, H. Shachnai, and T. Tamir *On chromatic sums and distributed resource allocation*. Information and Computation 140, 183–202, 1998.
2. A. Bar-Noy, G. Kortsarz *The minimum color-sum of bipartite graphs*. Journal of Algorithms 28, 339–365, 1998.

3. S. Cosares and I. Saniee *An Optimization Problem Related to Balancing Loads on SONET Rings*. Telecommunication Systems, Vol. 3, No. 2, 165–181, 1994.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman Publication, New York, 1979.
5. M. R. Garey, D. S. Johnson, G. L. Miller and C. H. Papadimitriou *The complexity of coloring circular arcs and chords*. SIAM Journal on Algebraic and Discrete Methods, 1(2), 216–227, 1980.
6. M. M. Halldorsson, G. Kortsarz and H. Shachnai *Minimizing Average Completion of Dedicated Tasks and Partially Ordered Sets*. Proc. of Fourth International Workshop on Approximation Algorithms (APPROX'01), Springer Verlag LNCS 2129, 114–126, 2001.
7. K. Jansen *Approximation Results for the Optimum Cost Chromatic Partition Problem*. Journal of Algorithms 34(1), 54–89, 2000.
8. S. Khanna *A Polynomial Time Approximation Scheme for the SONET Ring Loading Problem*. Bell Labs Technical Journal, Spring, 36–41, 1997.
9. L. G. Kroon, A. Sen, H. Deng and A. Roy *The optimal cost chromatic partition problem for trees and interval graphs*. Graph Theoretical Concepts in Computer Science WG 96, Como, LNCS, 1996.
10. E. Kubicka *The chromatic sum of a graph*. Ph.D. thesis, Western Michigan University, 1989.
11. E. Kubicka, G. Kubicki, and D. Kountanis. *Approximation Algorithms for the Chromatic Sum*. Proc. of the First Great Lakes Computer Science Conf., Springer LNCS 507, 15–21, 1989.
12. E. Kubicka and A. J. Schwenk *An introduction to chromatic sums*. Proceedings of the seventeenth Annual ACM Comp. Sci., Conf. ACM Press 39–45, 1989.
13. V. Kumar *Approximating circular arc coloring and bandwidth allocation in all-optical ring networks* Proc. 1st Int. Workshop on Approximation Algorithms for Combinatorial Problems, Lecture Notes in Comput. Sci., Springer-Verlag, 147–158, 1998.
14. Y. S. Myung *An Efficient Algorithm for the Ring Loading Problem with Integer Demand Splitting*. SIAM Journal on Discrete Mathematics, Volume 14, Number 3, 291–298, 2001.
15. S. Nicoloso, X. Song and M. Sarrafzadeh *On the sum coloring problem on interval graphs*. Algorithmica 23, 109–126, 1999.
16. A.E. Ozdaglar and D.P. Bertsekas *Routing and wavelength assignment in optical networks*. IEEE/ACM Transactions on Networking, Vol. 11, pp 259–272, April 2003.
17. J. Powers *An introduction to Fiber Optic Systems*. McGraw-Hill; 2nd edition, 1997.
18. R. Ramaswami and K.N. Sivarajan *Routing and wavelength assignment in all-optical networks* IEEE/ACM Transactions on Networking, Vol. 3, pp 489–499, Oct. 1995
19. A. Schrijver, P. Seymour, P. Winkler *The Ring Loading Problem*. SIAM Journal on Discrete Math., Vol. 11, 1–14, February 1998.
20. A. Sen, H. Deng and S. Guha *On a graph partition problem with an application to VLSI layout*. Information Processing Letters 24, 133–137, 1987.
21. K. J. Supowit *Finding a maximum planar subset of a set of nets in a channel*. IEEE Trans. on Computer Aided Design, CAD 6, 1, 93–94, 1987.
22. P. Winkler and L. Zhang *Wavelength Assignment and Generalized Interval Graph Coloring*. Proc. Symposium on Discrete Algorithms (SODA), pp. 830–831, 2003.

Lagrangian Relaxation for the k-Median Problem: New Insights and Continuity Properties

Aaron Archer*, Ranjithkumar Rajagopalan**, and David B. Shmoys***

School of Operations Research and Industrial Engineering, Cornell University
Ithaca, NY 14853 {aarcher, ranjith, shmoys}@cs.cornell.edu

Abstract. This work gives new insight into two well-known approximation algorithms for the uncapacitated facility location problem: the primal-dual algorithm of Jain & Vazirani, and an algorithm of Mettu & Plaxton. Our main result answers positively a question posed by Jain & Vazirani of whether their algorithm can be modified to attain a desired “continuity” property. This yields an upper bound of 3 on the integrality gap of the natural LP relaxation of the k -median problem, but our approach does not yield a polynomial time algorithm with this guarantee. We also give a new simple proof of the performance guarantee of the Mettu-Plaxton algorithm using LP duality, which suggests a minor modification of the algorithm that makes it Lagrangian-multiplier preserving.

1 Introduction

Facility location problems have been widely studied in both the operations research and computer science literature. We consider the two most popular variants of facility location: the k -median problem and the uncapacitated facility location problem (UFL). In both cases, we are given a set $\mathcal{C}$ of *clients* who must be served by a set $\mathcal{F}$ of *facilities*, and distances c_{ij} for all $i, j \in \mathcal{F} \cup \mathcal{C}$. When $i \in \mathcal{F}$ and $j \in \mathcal{C}$, c_{ij} is the cost of serving client j from facility i . We assume that these distances form a semi-metric; that is, $c_{ij} = c_{ji}$, and $c_{ik} \leq c_{ij} + c_{jk}$ for all $i, j, k \in \mathcal{F} \cup \mathcal{C}$. The goal is to open some subset of facilities $S \subseteq \mathcal{F}$ in order to minimize the total *connection cost* of serving each client from its closest facility, subject to some limitations on S . Whereas k -median imposes the hard constraint $|S| \leq k$, in UFL we have *facility costs* f_i for all $i \in \mathcal{F}$, and we aim to minimize the sum of the facility and connection costs.

Both problems are NP-hard, so we are interested in obtaining approximation algorithms. An α -*approximate* solution is one whose objective function is within a factor of α of the optimal solution. An α -*approximation* algorithm is one that runs in polynomial time and always returns an α -approximate solution. One primary theme of this line of research is to exploit a classical linear programming (LP) relaxation of the problem, initially proposed by Balinski [4]. We contribute to this vein by shedding new light on two existing UFL algorithms, the primal-dual algorithm of Jain & Vazirani (JV) [15], and the algorithm of Mettu & Plaxton (MP) [21].

* Supported by the Fannie and John Hertz Foundation and by NSF grant CCR-0113371.

** Research partially supported by NSF grant CCR-9912422.

*** Research partially supported by NSF grant CCR-9912422.

We show that the JV algorithm can be made "continuous," resolving a question posed in [15]. Because of their results connecting the k -median and UFL problems via Lagrangian relaxation, our result proves that the integrality gap of the most natural LP relaxation for k -median is at most 3, improving the previous best upper bound of 4 [7]. Since our algorithm involves solving the NP-hard maximum independent set problem, it does not lead directly to a polynomial-time 3-approximation algorithm; nonetheless, we believe that it is a significant step in that direction.

Mettu & Plaxton [21] prove that their algorithm achieves an approximation factor of 3, but their analysis never explicitly mentions an LP. Because the MP and JV algorithms appear superficially to be very similar and both achieve a factor of 3, many researchers wondered whether there was a deeper connection. We exhibit a dual solution that proves the MP primal solution is within a factor of 3 of the LP optimum. Interpreting their algorithm within an LP framework yields an additional benefit: it highlights that a slight modification of MP also satisfies the *Lagrangian-multiplier preserving* (LMP) property, which was not previously known. We note that Pál & Tardos independently constructed the same dual solution for use in creating cross-monotonic cost-sharing methods for facility location in a game-theoretic context [22].

The UFL problem has been studied from many perspectives since the 1960's, but the first approximation algorithm was given much later by Hochbaum [13], who achieved an $O(\log |C|)$ factor using a method based on greedy set cover. Shmoys, Tardos & Aardal [23], gave the first constant factor of 3.16. A series of papers have improved this to 1.52, by Mahdian, Ye & Zhang [19]. In the process, many and varied techniques have been brought to bear on the problem, and the insights gained have been applied elsewhere. Most prominent among the algorithmic and analytical techniques used have been LP rounding, filtering, various greedy algorithms, local search, primal-dual methods, cost-scaling, and dual fitting [7,9,12,14,15,17,23,24]. Guha & Khuller [12] showed that UFL cannot be approximated to a factor better than 1.463 unless $P = NP$.

K -median seems to be more difficult. The best hardness bound known is $1 + \frac{2}{e}$ [14], and the standard LP relaxation has an integrality gap of at least 2. Lin & Vitter [18] gave a constant-factor bicriterion approximation algorithm, and Bartal [5,6] achieved a near-logarithmic factor via probabilistic tree-embeddings, but the first constant factor of $6\frac{2}{3}$ was given by Charikar, Guha, Tardos & Shmoys [8], who used LP rounding. This factor was improved to 6 by Jain & Vazirani [15], 4 by Charikar & Guha [7], and $(3 + \epsilon)$ by Arya et al. [3]. The factor of 4 is attained via a refinement of the work of Jain & Vazirani, while the $(3 + \epsilon)$ is completely different, using local search.

Basic economic reasoning shows a connection between UFL and k -median. Consider a uniform facility cost z in the UFL. When $z = 0$, the best solution opens all facilities. As z increases from zero, the number of open facilities in the optimal solution decreases monotonically to one. Suppose some value of z causes the optimal UFL solution to open exactly k facilities S . Then S is also the optimal k -median solution.

Jain & Vazirani [15] exploit this relationship by interpreting the standard LP relaxation of UFL as the Lagrangian relaxation of the LP for k -median. Their elegant primal-dual UFL algorithm achieves a guarantee of 3, and also satisfies the LMP property. They then show how to convert any LMP algorithm into an approximation algorithm for k -median while losing an additional factor of 2 in the guarantee. More importantly

for us, they show that the solution S output by their UFL algorithm is a 3-approximate solution for the $|S|$ -median problem. Thus, if one can find, in polynomial time, a value of z such that the JV algorithm opens exactly k facilities, this constitutes a 3-approximation algorithm for the k -median problem. Sadly, there are inputs for which no value of z causes the JV algorithm (as originally stated) to open exactly k facilities.

We modify the JV algorithm to attain the following *continuity* property. Consider the solution $S(z)$ output by the algorithm, as a function of the uniform facility cost z . As z changes, we ensure that $|S(z)|$ never jumps by more than 1. Since the algorithm opens all facilities when $z = 0$ and only one when z is sufficiently large, there is some value for which it opens exactly k . By standard methods (either binary search or Megiddo's parametric search [20]), we can find the desired value using a polynomial number of calls with different values of z . This appears to answer the question posed in [15]. Unfortunately, our algorithm involves finding maximum independent sets, which is NP-hard. This leaves the open question of whether one can achieve a version of JV that has the continuity property *and* runs in polynomial time.

There are two rays of light. First, our algorithm does prove that the integrality gap of the standard k -median LP is at most 3. This was not known before, and it is novel because most proofs that place upper bounds on the integrality gaps of LP relaxations rely on polynomial-time algorithms. (For an interesting exception to this rule, see [2].) Second, it is enough to compute maximal independent sets that are continuous with respect to certain perturbations of the graph¹. The only types of sets that we know to be continuous are maximum independent sets, but we are hopeful that one could compute, in polynomial time, some other type of continuous maximal independent set.

2 Ensuring Continuity in Jain-Vazirani

The JV algorithm for UFL is based on the following standard LP relaxation for the problem (originally proposed by Balinski [4]), and its dual.

Primal LP:

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{F}} f_i y_i + \sum_{ij \in \mathcal{F} \times \mathcal{C}} c_{ij} x_{ij} \\ \text{such that:} \quad & \sum_{i \in \mathcal{F}} x_{ij} = 1 \quad \forall j \in \mathcal{C} \\ & y_i - x_{ij} \geq 0 \quad \forall ij \in \mathcal{F} \times \mathcal{C} \\ & y_i, x_{ij} \geq 0 \quad \forall ij \in \mathcal{F} \times \mathcal{C} \end{aligned}$$

Dual LP:

$$\begin{aligned} \max \quad & \sum_{j \in \mathcal{C}} v_j \\ \text{such that:} \quad & \sum_{j \in \mathcal{C}} w_{ij} \leq f_i \quad \forall i \in \mathcal{F} \\ & v_j - w_{ij} \leq c_{ij} \quad \forall ij \in \mathcal{F} \times \mathcal{C} \\ & v_j, w_{ij} \geq 0 \quad \forall ij \in \mathcal{F} \times \mathcal{C} \end{aligned}$$

Adding constraints $y_i, x_{ij} \in \{0, 1\}$ gives an exact IP formulation. The variable y_i indicates whether facility i is open, and x_{ij} says whether client j is connected to facility i . Intuitively, v_j is the total amount of money that client j is willing to pay to be served: w_{ij} is its share towards the cost of facility i , and the rest pays for its connection cost.

The JV algorithm operates in two phases. Phase I consists of growing the dual variables, maintaining dual feasibility, and gradually building a primal solution until

¹ The class of perturbations needs to be defined carefully; an earlier version of this abstract proposed one that was more general than necessary, and implied that the only possible realization of this approach required a maximum independent set.

that solution is feasible. Phase II is a cleanup phase in which we keep only a subset of the facilities opened in phase I. This results in the following theorem.

Theorem 1 (Jain-Vazirani 2001) *The Jain-Vazirani facility location algorithm yields a feasible integer primal solution and a feasible dual solution to the UFL LP, satisfying $C + 3F \leq 3 \sum_{j \in \mathcal{C}} v_j \leq 3OPT$, where OPT is the value of the optimal UFL solution.*

We now describe the algorithm precisely but conceptually, motivating each step but ignoring the implementation details. We envision dual and primal solutions changing over time. At time zero, we set all primal and dual variables to zero, so the dual is feasible and the primal is infeasible. Throughout phase I, we maintain dual feasibility and work towards primal feasibility. We also enforce primal complementary slackness, meaning that we never open a facility i unless it is fully paid for by the dual variables (i.e., $\sum_j w_{ij} = f_i$) and we connect client j to facility i only if $v_j = c_{ij} + w_{ij}$, i.e., j 's dual variable fully pays for connection cost and its share of facility i 's cost.

We initially designate all clients as *active*, and raise their dual variables at unit rate. Eventually, some edge ij goes *tight*, meaning that $v_j = c_{ij}$, i.e., client j 's dual variable has completely paid for its connection cost to facility i . We continue raising the v_j variables at unit rate for all active clients j , but now we must also raise the w_{ij} cost shares for all tight edges ij . Eventually, we pay for and open some facility i when the constraint $\sum_j w_{ij} \leq f_i$ goes tight. Now we must freeze all of the cost shares w_{ij} in order to maintain dual feasibility, so we must also freeze the dual variable v_j for every client j with a tight edge to facility i . Fortunately, facility i is now open, so we can assign client j to be served by facility i and declare it *inactive*. We refer to facility i as client j 's *connecting witness*. Conveniently, v_j exactly pays for j 's connection cost plus its share of facility i 's cost, since $v_j = c_{ij} + w_{ij}$. We continue in this manner. It can also occur that an active client gains a tight edge to a facility that is already open. In this case, the client is immediately connected to that facility. Phase I terminates when the last active client is connected. If any combination of events is set to occur simultaneously, we can break ties in an arbitrary order. Notice that the tiebreaking rule has no effect on the dual solution generated.

At the end of phase I, we have some preliminary set S_0 of open facilities. As we have mentioned, the algorithm opens a facility only when the w_{ij} variables fully pay for it, and the v_j variable for client j exactly pays for its connection cost plus its share of the facility cost for its connecting witness. Then why is S_0 not an optimal solution? It is because some client may have contributed a non-zero cost share to some open facility to which it is not connected. Thus, we must clean up the solution to avoid this problem. In phase II, we select a subset of facilities $S \subseteq S_0$ so that each client pays a positive cost share to at most one open facility. Every client that has a tight edge to a facility in S is said to be *directly connected*. Thus, the directly connected clients exactly pay for their own connection costs and all of the facility costs. The trick is, each client j that is not directly connected must still be connected to some facility i . We obtain a 3-approximation algorithm if we can guarantee that $c_{ij} \leq 3v_j$.

Phase II proceeds as follows. We construct a graph G with vertices S_0 , and include an edge between $i, k \in S_0$ if there exists some client j such that $w_{ij}, w_{kj} > 0$. We must select S to be an independent set in G . Otherwise, some client j offered cost shares to

two facilities in S , but it can afford to pay for only one. We might as well choose S to be a *maximal* independent set (meaning that no superset of S is also an independent set, so every vertex in $S_0 - S$ is adjacent to a vertex in S). For each client j that is not directly connected, consider its connecting witness i . Since $i \notin S$, there must exist an adjacent facility $k \in S$, so we connect j to k . This completes the description of the algorithm.

In their original paper [16], Jain & Vazirani chose a particular set S , but in the journal version, they modify their analysis to accommodate any maximal independent set. Later, we will choose a *maximum* (cardinality) independent set, but for Theorem 1 and the present discussion, any maximal independent set suffices.

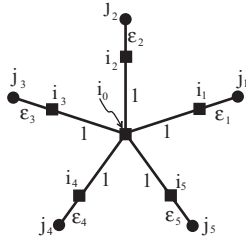
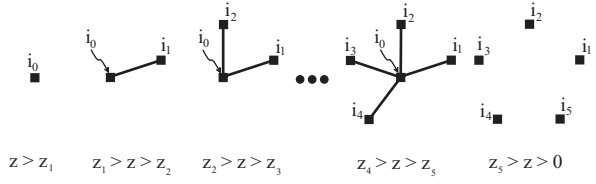
The LMP property becomes important when we view the LP relaxation of UFL as the Lagrangian relaxation of the standard LP relaxation of k -median. The k -median LP is the same as the UFL LP, except there is no facility cost term in the objective, and we add the constraint $\sum_i y_i \leq k$. By Lagrangian relaxation, we mean to remove the cardinality constraint, set a non-negative penalty parameter z , and add the term $z(\sum_i y_i - k)$ to the objective function. This penalizes solutions that violate the constraint by opening more than k facilities, and gives a bonus to solutions that open fewer than k . Aside from the constant term of $-zk$, this is precisely the same as the LP relaxation of UFL, setting all facility costs to z . Notice that the objective function matches the true k -median objective whenever exactly k facilities are opened. Thus, every feasible solution for the original k -median LP is also feasible for its Lagrangian relaxation, and the objective function value in the relaxation is no greater than in the original LP. Therefore, every dual feasible solution for the Lagrangian relaxation provides a lower bound on the optimal k -median solution. These observations lead to the following result in [15].

Theorem 2 *Suppose that we set all facility costs to $z > 0$, so that the JV algorithm opens exactly k facilities. Then this is a 3-approximate k -median solution.*

A bad example and how to fix it in general. We first give a well-known example showing that the JV algorithm as described above does not satisfy the continuity property. We then show that perturbing the input fixes this bad example. Our main result shows that this trick works in general. Consider the metric space given by a star with h arms, each of length 1. At the end of each arm there is one client j and one potential facility i_j . There is also one facility (called i_0) located at the hub of the star. (See Figure 1, setting all $\epsilon_j = 0$.) When $z < 1 + \frac{1}{h-1}$, each client completely pays for the facility located on top of it by time z , while the hub facility has still not been paid for. Hence, $G(z)$ consists of these h facilities, with no edges between them. When $z > 1 + \frac{1}{h-1}$, the hub is opened and all clients connected to it before time z , so $G(z)$ has just one vertex, the hub. Thus, $|S(z)|$ jumps from h down to 1 at the critical value $z = 1 + \frac{1}{h-1}$.

Now perturb the instance by an arbitrarily small amount, moving each client j out past its nearby facility by an amount $\epsilon_j \ll 1$, where $0 = \epsilon_1 < \dots < \epsilon_h$. Let $\epsilon = \sum_{j=1}^h \epsilon_j$, and let $z_1 = \frac{h+\epsilon}{h-1}$. For $z > z_1$, the hub facility is opened before any of the arm facilities, so $G(z)$ is just one isolated vertex. At the critical value $z = z_1$, the hub facility is paid for at exactly the same moment as facility 1. For slightly smaller values of z , facility 1 is paid for first, then the hub is opened before any other facility is paid for. Clearly, there exist some $z_1 > z_2 > \dots > z_h > 0$ such that when $z \in (z_{i+1}, z_i)$, facilities $1, \dots, i$ are opened before the hub in phase I, and facilities $(i+1), \dots, h$ are not opened. For z

Discontinuity Example

 $G(z)$ for varying values of z **Fig. 1.** Discontinuity example (with $h = 5$) and its perturbation.

in this range, $G(z)$ consists of the hub facility with edges to facilities $1, \dots, i$, because client j contributes toward the costs of both the hub and the open facility j , for $1 \leq j \leq i$. For $z \in [0, z_h]$, $G(z)$ contains just isolated vertices $i_1, \dots, i_h$.

Theorem 1 holds no matter which maximal independent set we choose in phase II, so let $S(z)$ be a maximum independent set. When $z \in (z_{i+1}, z_i)$, $S(z)$ consists of the i facilities $1, \dots, i$. Thus, $|S(z)|$ changes by at most one at each of the critical values. We have made JV algorithm continuous by perturbing the input an arbitrarily small amount. Our main result is that this trick always works.

We now give some definitions to make our claim precise. We also state our two main results, Theorems 3 and 4, but prove them later. An *event* of the algorithm is the occurrence that either an edge ij goes tight (because client j is active at time c_{ij}) or some facility becomes paid for in phase I. We say that an instance of the UFL problem is *degenerate* if there is some time at which three or more events coincide, or there are at least two points in time where two events coincide. An instance of the k -median problem is degenerate if there exists some $z > 0$ that yields a degenerate UFL instance. (For every non-trivial instance, it is easy to select z so that there is *one* time when two events coincide.) Notice that an instance of the k -median problem simply consists of the distances $\{c_{ij} : ij \in \mathcal{F} \times \mathcal{C}\}$, so we consider an instance to be a point in $\mathbb{R}_+^{\mathcal{F} \times \mathcal{C}}$.

Theorem 3 *The set of all degenerate instances to the k -median problem has Lebesgue measure zero.*

For a non-degenerate UFL instance, let us define the *trace* of the algorithm to be the sequence of events encountered during phase I. Notice that $G(z)$ (and consequently, $|S(z)|$) depends only on the trace. Define z_0 to be a *critical value* of z if, when $z = z_0$, there is some point in time where at least two events coincide. For a graph G , let $\mathcal{I}(G)$ denote the size of the largest independent set in G .

Theorem 4 *As z passes through a critical value at which only two events coincide, $\mathcal{I}(G(z))$ changes by at most 1.*

As we will show, this holds because $G(z)$ changes only slightly when z passes through a non-degenerate critical value. Thus, the algorithm is continuous if our k -median instance is non-degenerate.

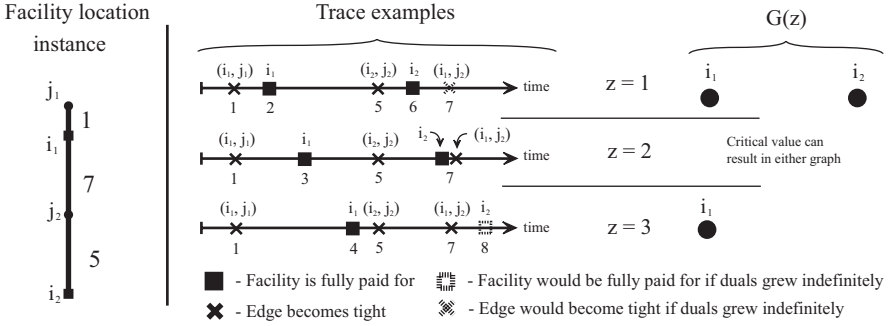


Fig. 2. Trace example.

Example of traces. To clarify the concept of a trace, we give three traces for the simple facility location instance in Figure 2. When $z = 1$, both i_1 and i_2 are opened, j_1 is connected to i_1 and j_2 is connected to i_2 . The edge (i_1, j_2) would become tight at time 7 if v_{j_2} were allowed to grow indefinitely, but v_{j_2} stops growing at time 6, when j_2 is connected to i_2 . When $z = 3$, j_1 pays to open i_1 and j_2 connects to it before i_2 is paid for. The figure shows that i_2 would have opened at time 8 if v_{j_3} were allowed to continue growing. At the critical value $z = 2$, i_2 is paid for at the same time that (i_1, j_2) becomes tight, so tiebreaking determines which of the previous solutions is output. The final output of the algorithm depends only on the order of events, not on the actual times. Thus, as z changes, events may slide forward and backward on the trace, but the output changes only at critical values, when events change places.

Exploiting non-degeneracy. For a non-degenerate instance of k -median, we wish to understand how $G(z)$ changes when z passes through a critical value, as summarized in the following theorem.

Theorem 5 *When z passes through a critical value where exactly two events coincide, the graph $G(z)$ can change only in one of the following ways: (a) a single existing facility is deleted (along with its incident edges), (b) a single new facility is added, along with edges to one or more cliques of existing facilities, (c) a single existing facility gains edges to one clique of facilities, or loses edges to one clique.*

Proof: We need to determine how overlapping events can change $G(z)$ at a critical value z . To this end, we define one more graph, $H(z)$, which has one node per client, one node per facility opened in phase I, and an edge between every client j and facility i such that $w_{ij} > 0$. Thus, the edges of $G(z)$ connect facilities for which there exists a two-hop path in $H(z)$. We prove that, at a critical value of z , $H(z)$ can change only by addition or deletion of one facility (along with its incident edges), or by addition or deletion of a single client-facility edge. The theorem follows.

Given the order of events, we determine the edges of $H(z)$ as follows. For each client j and open facility i , $H(z)$ includes an edge if the edge event (i, j) occurred strictly before the facility event i . Since each v_j increases at unit rate from time $t = 0$, then

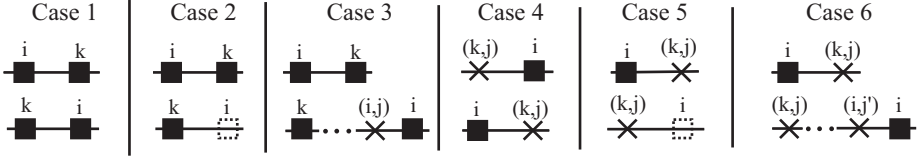


Fig. 3. Trace change cases.

stops when the client j is connected, the edge event for (i, j) will either occur at $t = c_{ij}$, or not at all if the client is connected before that time. Facility events, on the other hand, change position depending on z . However, if there is a facility event for a certain value of z , that event will disappear as z changes only if it gets moved past the time where all clients are connected. Thus, the graph $H(z)$ changes in a restricted way. The vertex set changes only if a facility event is added or removed from the trace. The presence of edge (i, j) changes only if facility event i and edge event (i, j) change their relative order. Critical values of z fall into several cases, as shown in Figure 3. For ease of exposition, we refer to the top trace occurring “before” the change in z , and the bottom trace “after.”

Case 1: Facilities i and k swap places. This can happen if different numbers of clients are contributing to the two facilities, causing different rates of payment. Here, the set of open facilities remains the same, and the positions of edge events relative to i and k remain the same, so $H(z)$ does not change.

Case 2: Facility i disappears when k opens first. This happens if all clients that were paying for i connect to k when it opens, and no other clients go tight to i before the end of phase I, so i remains unopened. The relative order of events remains intact, except that i is removed, so $H(z)$ changes by removal of i and all incident edges.

Case 3: Facility i jumps later in time when k opens first. Similar to case 2, this happens if all clients that were paying for i instead connect to k when it opens, causing i to remain closed for a period of time, until the next client j grows its dual enough to go tight and finish paying for i , possibly much later in the trace. Here, $H(z)$ gets one new edge (i, j) .

Case 4: Facility i moves across edge (k, j) . If $i = k$, then the order of the two events determines whether j has strictly positive cost share to i . Thus, as the facility event moves to the left, $H(z)$ loses the edge (i, j) . If $i \neq k$, then $H(z)$ does not change, because the order of the edge event (k, j) and the facility event k (if it exists) is preserved.

Case 5: Facility i disappears as it crosses edge event (k, j) to the right (where $k \neq i$). Similar to case 2, this happens if j is the only client contributing to i , but stops when it connects to an open facility k . As in case 2, i gets deleted from $H(z)$.

Case 6: Facility i jumps later in time when the edge event (k, j) occurs before it ($k \neq i$). Similar to case 3, this happens if j is the only client contributing to i , but stops when it connects to k . However, i is opened later as some other client j' becomes tight and pays for the excess. Here, $H(z)$ gets one new edge (i, j') . ■

Clearly, the types of graph perturbations described in Theorem 5 change $\mathcal{I}(G(z))$ by at most one, which proves Theorem 4. By definition, non-degenerate k -median instances

are ones where we can apply Theorem 4 at every critical value, so our algorithm is continuous when applied to these instances.

Attaining non-degeneracy.. Our last task is to prove Theorem 3. Our approach is to view UFL instances (c, z) with uniform facility costs z as points in $\mathbb{R}_+^{\mathcal{F} \times \mathcal{C}} \times \mathbb{R}_+$, i.e., the positive orthant of $(N + 1)$ -dimensional space, where $N = |\mathcal{F}| \cdot |\mathcal{C}|$. Each possible trace corresponds to a region of space consisting of the UFL instances that result in this trace. A k -median instance with cost vector c is represented by the ray $\{(c, z) : z > 0\}$. As long as this ray passes through no degenerate UFL points, then the k -median instance c is non-degenerate. In other words, the set of all degenerate k -median instances is simply the projection onto the $z = 0$ plane of the set of all degenerate UFL instances. Theorem 3 relies on the following result.

Theorem 6 *Each possible trace corresponds to a region of (c, z) -space bounded by a finite number of hyperplanes.*

We include detailed proofs of Theorems 6 and 3 in the full version of this paper. The crux is that every degenerate UFL instance lies at the intersection of two hyperplanes, hence on one of a finite number of $(N - 1)$ -dimensional planes. The same goes for the projection, which thus has zero Lebesgue measure in $\mathbb{R}_+^N$.

3 Facility Location Algorithm of Mettu and Plaxton

So far we have been considering the UFL algorithm of Jain & Vazirani because it has the LMP property. We now turn to a similar algorithm proposed by Mettu & Plaxton (MP). In its original form, it does not have the LMP property. However, using an LP-based analysis, we show that a slightly modified version of this algorithm attains the LMP property while delivering the same approximation factor.

Algorithm Description. The MP algorithm associates a ball of clients with each facility, and then chooses facilities in a greedy fashion, while preventing overlapping balls. Define the radii $r_i : i \in \mathcal{F}$, so that $f_i = \sum_{j \in \mathcal{C}} \max(0, r_i - c_{ij})$. Intuitively, these radii represent a sharing of the facility cost among clients. If each client in a ball of radius r_i around facility i pays a total of r_i , that will pay for the connection costs in the ball, as well as the facility cost f_i . Without loss of generality, let $r_1 \leq r_2 \leq \dots \leq r_n$. Let B_i be the ball of radius r_i around i . In ascending order of radius, include i in the set of open facilities if there are no facilities within $2r_i$ already open. The algorithm ensures that the balls around open facilities are disjoint, so no client lies in the balls of two different open facilities. Thus, each client contributes to at most one facility cost.

Proof of Approximation Factor. We now use LP duality to prove that MP achieves an approximation factor of 3. We also prove that a slightly modified algorithm MP- β has the LMP property. The algorithm MP- β is MP with one modification. In MP- β , choose the radii r_i so that $\beta f_i = \sum_{j \in \mathcal{C}} \max(0, r_i - c_{ij})$. Our analysis uses the same LP formulation as before.

Theorem 7 *MP- β delivers a 3-approximate solution to the facility location problem for $1 \leq \beta \leq \frac{3}{2}$. Furthermore, if F is the facility cost of the algorithm's solution, C is the algorithm's connection cost, and OPT is the optimal solution cost, then $C + 2\beta F \leq \sum_j v_j \leq 3OPT$.*

We prove this result by exhibiting a particular feasible dual solution. Let Z be the set of facilities opened by MP- β , and let $r_i : i \in \mathcal{F}$ be the radii used. We need to construct a set of v_j and w_{ij} from this solution. Set $w_{ij} = \frac{1}{\beta} \max(0, r_i - c_{ij})$ for $ij \in \mathcal{F} \times \mathcal{C}$. Say that j contributes to i if $w_{ij} > 0$. Then, set $v_j = \min_{i \in \mathcal{F}} c_{ij} + w_{ij}$. It is clear that the v and w vectors are non-negative. By the choice of the vector v , we automatically satisfy $v_j - w_{ij} \leq c_{ij}, \forall ij \in \mathcal{F} \times \mathcal{C}$. Finally, r_i and w_{ij} were chosen so that $\beta f_i = \sum_{j \in \mathcal{C}} \max(0, r_i - c_{ij}) = \beta \sum_{j \in \mathcal{C}} w_{ij}$. Thus, our dual solution is feasible.

It remains to be shown that $\sum_{j \in \mathcal{C}} d(j, Z) + 2\beta \sum_{i \in Z} f_i \leq 3 \sum_{j \in \mathcal{C}} v_j$. We will show that each $3v_j$ pays to connect j to some open facility i , and also pays for 2β times j 's cost share (if one exists). Define $s_j = w_{ij}$ if there is an i such that i is open and $w_{ij} > 0$, and set $s_j = 0$ otherwise. Note that this is well defined because j can be in at most one open facility's ball. Since $f_i = \sum_{j \in \mathcal{C}} w_{ij}$, $\sum_{i \in Z} f_i = \sum_{j \in \mathcal{C}} s_j$. Furthermore, $\forall i \in Z, d(j, Z) \geq c_{ji}$ by definition. Thus, in order to show $3 \sum_{j \in \mathcal{C}} v_j \geq 2\beta \sum_{i \in Z} f_i + \sum_{j \in \mathcal{C}} d(j, Z)$, it is enough to show that for all $j \in \mathcal{C}$ there exists $i \in Z$ such that $3v_j \geq c_{ij} + 2\beta s_j$.

Call the facility i that determines the minimum in $\min_{i \in \mathcal{F}} c_{ij} + w_{ij}$ the *bottleneck* of j . The proof of Theorem 7 relies on some case analysis, based on the bottleneck of j . Before we analyze the cases, we need four lemmas, stated here without proof.

Lemma 1. *For any facility $i \in \mathcal{F}$ and client $j \in \mathcal{C}$, $r_i \leq c_{ij} + \beta w_{ij}$.*

Lemma 2. *If $\beta \leq \frac{3}{2}$, and i is a bottleneck for j , then $3v_j \geq 2r_i$.*

Lemma 3. *If an open facility i is a bottleneck for j , then j cannot contribute to any other open facility.*

Lemma 4. *If a closed facility i is a bottleneck for j and k is the open facility that caused i to close, then $c_{kj} \leq \max(3, 2\beta)v_j$.*

Now we prove the theorem in cases, according to the bottleneck for each client j .

Proof of Theorem 7: We must show for all j that there is some open facility i such that $3v_j \geq c_{ij} + 2\beta w_{ij}$. Consider the bottleneck of an arbitrary client j .

Case 1: The bottleneck is some open facility i . By Lemma 3, we know that j cannot contribute to any other open facility. So connect j to facility i . If $c_{ij} < r_i$ then $0 < w_{ij} = s_j$ and $v_j = c_{ij} + s_j$. Thus, v_j pays exactly for connection cost and the cost share. If $c_{ij} \geq r_i$, we know that $s_j = 0$, since $w_{ij} = 0$, and j cannot contribute to any other facility. So $v_j = c_{ij}$. Thus, v_j pays exactly for connection cost, and there is no cost share.

Case 2: The bottleneck is some closed facility i , and j does not contribute to any open facility. We know $s_j = 0$ since j does not contribute to any open facility. We also know

there is some open facility k that caused i to close. Connect j to k . By Lemma 4, we know that $c_{kj} \leq \max(3, 2\beta)v_j$. Since $\beta \leq \frac{3}{2}$, we have that $3v_j \geq c_{kj}$. Thus, $3v_j$ pays for the connection cost, and there is no cost share.

Case 3: The bottleneck is some closed facility i , and there is some open facility l with $w_{lj} > 0$, and l was not the reason that i closed. Since $w_{lj} > 0$, $s_j = w_{lj}$. Connect j to l incurring $c_{lj} + w_{lj}$. Since $w_{lj} = s_j$, we have that $c_{lj} + \beta s_j = r_l$. Since k and l are both open, we have that $c_{lk} \geq 2r_l$. Using the triangle inequality, this gives $2c_{lj} + 2\beta s_j \leq c_{lk} \leq c_{lj} + c_{kj}$, or $c_{lj} + 2\beta s_j \leq c_{kj}$. Just as in Case 2, we know there is some open facility $k \neq l$ that prevented i from opening, which means $c_{ik} \leq 2r_i$. By Lemma 4, we know $c_{kj} \leq 3v_j$. So, putting it all together, we have $c_{lj} + 2\beta s_j \leq c_{kj} \leq 3v_j$. Thus, $3v_j$ pays for the connection cost, and 2β times the cost share.

Case 4: The bottleneck is some closed facility i and there is some open facility k with $w_{kj} > 0$ and k caused i to be closed. Here, $s_j = w_{kj}$. From Lemma 2, we know that $3v_j \geq 2r_i$. Since k caused i to close, $r_i \geq r_k = c_{kj} + \beta s_j$. Thus, we have $3v_j \geq 2r_i \geq 2c_{kj} + 2\beta s_j \geq c_{kj} + 2\beta s_j$. So $3v_j$ pays for the connection cost and 2β times the cost share.

Thus, in each case, we have shown that there is an open facility i that satisfies $3v_j \geq c_{ij} + 2\beta s_j$ which shows that the algorithm delivers a solution that satisfies $C + 2\beta F \leq 3OPT$, giving a 3-approximation so long as $\beta \geq \frac{1}{2}$. ■

4 Final Thoughts

The preceding theorem shows that the algorithm $MP-\frac{3}{2}$ has the LMP property necessary to build a k -median algorithm. The primary benefit of using $MP-\frac{3}{2}$ instead of another LMP algorithm with guarantee 3 is the running time. The k -median approximation algorithm runs the facility location algorithm several times as a black box. Whereas the original JV facility location algorithm had a running time of $O(|\mathcal{F}||\mathcal{C}| \log |\mathcal{F}||\mathcal{C}|)$, the algorithm $MP-\frac{3}{2}$ can be implemented to run in $O(|\mathcal{F}|^2 + |\mathcal{F}||\mathcal{C}|)$ time.

Any LMP algorithm with guarantee c that also has the continuity property analogous to Theorem 4 immediately yields a c -approximation for the k -median problem, because we can simply search for a value of z for which we open exactly k facilities. Unfortunately $MP-\frac{3}{2}$ is not continuous. We include an example demonstrating this fact in the full version of this paper. The tightest LMP result is the dual fitting algorithm of [14], which yields a factor of 2. However, on the star instance of Figure 1, this algorithm jumps from opening 1 facility to opening h of them at $z = \frac{h+\epsilon}{h-1}$. Thus, our modification of JV is the only LMP algorithm so far that has this property.

An important direction for future research is to identify a rule for computing maximal independent sets in polynomial time that satisfy the continuity property of Theorem 4, with $\mathcal{I}(G(z))$ replaced by $|S(z)|$. This would convert our existential result into a polynomial time 3-approximation algorithm for the k -median problem.

One algorithmic consequence of Theorem 3 is that we can always make an arbitrarily small perturbation to our given instance to transform it into a non-degenerate instance. However, for purposes of applying Theorems 4 and 5, it suffices to process trace changes one at a time for degenerate values of z . These same techniques can be applied to prove

analogous theorems about degeneracy in the prize-collecting Steiner tree algorithm of Goemans & Williamson [11], the other major example where Lagrangian relaxation has been used in approximation algorithms [10].

References

1. A. Ageev & M. Sviridenko. An approximation algorithm for the uncapacitated facility location problem. Manuscript, 1997.
2. S. Arora, B. Bollobas, & L. Lovász. Proving integrality gaps without knowing the linear program. *43rd FOCS*, 313–322, 2002.
3. V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, & V. Pandit. Local search heuristic for k -median and facility location problems. *33rd STOC*, 21–29, 2001.
4. M. Balinski. On finding integer solutions to linear programs. In *Proc. IBM Scientific Computing Symp. on Combinatorial Problems*, 225–248, 1966.
5. Y. Bartal. On approximating arbitrary metrics by tree metrics. *30th STOC*, 161–168, 1998.
6. Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. *37th FOCS*, 184–193, 1996.
7. M. Charikar & S. Guha. Improved combinatorial algorithms for the facility location and k -median problems. *40th FOCS*, 378–388, 1999.
8. M. Charikar, S. Guha, E. Tardos, & D.B. Shmoys. A constant-factor approximation algorithm for the k -median problem. *31st STOC*, 1–10, 1999.
9. F. Chudak & D.B. Shmoys. Improved approximation algorithms for uncapacitated facility location. *SIAM J. Comput.*, to appear.
10. F. Chudak, T. Roughgarden, & D.P. Williamson. Approximate k -MSTs and k -Steiner trees via the primal-dual method and Lagrangean relaxation. *8th IPCO, LNCS 2337*, 66–70, 2001.
11. M. Goemans & D.P. Williamson. A general approximation technique for constrained forest problems. *SICOMP* 24, 296–317, 1995.
12. S. Guha & S. Khuller. Greedy strikes back: improved facility location algorithms. *J. Alg.* 31, 228–248, 1999.
13. D. Hochbaum. Heuristics for the fixed cost median problem. *Math. Prog.* 22, 148–162, 1982.
14. K. Jain, M. Mahdian, E. Markakis, A. Saberi, & V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP To appear in *JACM*
15. K. Jain & V. Vazirani. Approximation algorithms for metric facility location and k -median problems using primal-dual schema and Lagrangian relaxation. *JACM* 48, 274–296, 2001.
16. K. Jain & V. Vazirani. Primal-dual approximation algorithms for metric facility location and k -median problems. *40th FOCS*, 2–13, 1999.
17. M. Korupolu, C.G. Plaxton, & R. Rajaraman. Analysis of a local search heuristic for facility location problems. *J. Alg.* 37, 146–188, 2000.
18. J.H. Lin & J. Vitter. Approximation algorithms for geometric median problems. *IPL* 44, 245–249, 1992.
19. M. Mahdian, Y. Ye, & J. Zhang. Improved approximation algorithms for metric facility location problems. *4th APPROX, LNCS 2462*, 229–242, 2002.
20. N. Meggido. Combinatorial optimization with rational objective functions. *Math. OR*, 4:414–424, 1979.
21. R. Mettu & C.G. Plaxton. The online median problem. *41st FOCS*, 339–348, 2000.
22. M. Pál & Éva Tardos. Strategy proof mechanisms via primal-dual algorithms. To appear in *44th FOCS*, 2003.
23. D.B. Shmoys, É. Tardos, & K. Aardal. Approximation algorithms for facility location problems. *29th STOC*, 265–274, 1997.
24. M. Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. *9th IPCO, LNCS 2337*, 240–257, 2002.

Scheduling for Flow-Time with Admission Control^{*}

Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Kedar Dhamdhere

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213,
USA. {nikhil,avrim,shuchi,kedar}@cs.cmu.edu

Abstract. We consider the problem of scheduling jobs on a single machine with preemption, when the server is allowed to reject jobs at some penalty. We consider minimizing two objectives: total flow time and total job-idle time (the idle time of a job is the flow time minus the processing time). We give 2-competitive online algorithms for the two objectives and extend some of our results to the case of weighted flow time and machines with varying speeds. We also give a resource augmentation result for the case of arbitrary penalties achieving a competitive ratio of $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ using a $(1 + \epsilon)$ speed processor. Finally, we present a number of lower bounds for both the case of uniform and arbitrary penalties.

1 Introduction

Consider a large distributed system with multiple machines and multiple users who submit jobs to these machines. The users want their jobs to be completed as quickly as possible, but they may not have exact knowledge of the current loads of the processors, or the jobs submitted by other users in the past or near future. However, let us assume that each user has a rough estimate of the typical time she should expect to wait for a job to be completed. One natural approach to such a scenario is that when a user submits a job to a machine, she informs the machine of her estimate of the waiting time if she were to send the job elsewhere. We call this quantity the *penalty* of a job. The machine then might service the job, in which case the cost to the user is the flow time of the job (the time elapsed since the job was submitted). Or else the machine might reject the job, possibly after the job has been sitting on its queue for some time, in which case the cost to the user is the penalty of the job plus the time spent by the user waiting on this machine so far.

To take a more human example, instead of users and processors, consider journal editors and referees. When an editor sends a paper to a referee, ideally she would like a report within some reasonable amount of time. Less ideally, she would like an immediate response that the referee is too busy to do it. But even worse is a response of this sort that comes 6 months later after the referee

^{*} This research was supported in part by NSF grants CCR-0105488, NSF-ITR CCR-0122581, NSF-ITR IIS-0121678, and an IBM Graduate Fellowship.

originally agreed to do the report. However, from the referee’s point of view, it might be that he thought he would have time when he received the request, but then a large number of other tasks arrived and saying no to the report (or to some other task) is needed to cut his losses.

Motivated by these scenarios, in this paper we consider this problem from the point of view of a single machine (or researcher/referee) that wants to be a good sport and minimize the total cost to users submitting jobs to that machine. That is, it wants to minimize the total time jobs spend on its “to-do list” (flow time) plus rejection penalties.¹

Specifically, we consider the problem of scheduling on a single machine to minimize flow time (also job-idle time) when jobs can be rejected at some cost. Each job j has a release time r_j , a processing time p_j , and we may at any time cancel a job at cost c_j . For most of the paper, we focus on the special case that the cancellation costs are all equal to some fixed value c — even this case turns out to be nontrivial — though we give some results for general c_j as well. In this paper, we consider clairvoyant algorithms, that is, whenever a job is released, its size and penalty is revealed.

In the flow-time measure, we pay for the total time a job is in the system. So, if a job arrives at time 1 and we finish it by time 7, we pay 6 units. If we choose to cancel a job, the cancellation cost is added on. Flow-time is equivalent to saying that at each time step, we pay for the number of jobs currently in the system (i.e., the current size of the machine’s to-do list). In the job-idle time measure, we pay at each time step for the number of jobs currently in the system minus one (the one we are currently working on), or zero if there are no jobs in the system. Because job idle time is smaller than flow time, it is a strictly harder problem to approximate, and can even be zero if jobs are sufficiently well-spaced. Preemption is allowed, so we can think of the processor as deciding at each time step how it wants to best use the next unit of time. Note that for the flow-time measure, we can right away reject jobs that have size more than c , because if scheduled, these add at least c to the flow-time. However, this is not true for the job-idle time measure.

To get a feel for this problem, notice that we can model the classic ski-rental problem as follows. Two unit-size jobs arrive at time 0. Then, at each time step, another unit-size job arrives. If the process continues for less than c time units, the optimal solution is not to reject any job. However, if it continues for c or more time units, then it would be optimal to reject one of the two jobs at the start. In fact, this example immediately gives a factor 2 lower bound for deterministic algorithms for job-idle time, and a factor $3/2$ lower bound for flow time.

To get a further feel for the problem, consider the following online algorithm that one might expect to be constant-competitive, but in fact does not work: Schedule jobs using the Shortest Remaining Processing Time (SRPT) policy (the optimal algorithm when rejections are not allowed), but whenever a job has been in the system for more than c time units, reject this job, incurring an additional

¹ However, to be clear, we are ignoring issues such as what effect some scheduling policy might have on the rest of the system, or how the users ought to behave, etc.

c cost. Now consider the behavior of this algorithm on the following input: m unit size jobs arrive at time 0, where $m < c$, and subsequently one unit size job arrives in every time step for n steps. SRPT (breaking ties in favor of jobs arriving earlier) will schedule every job within m time units of its arrival. Thus, the proposed algorithm does not reject any job, incurring a cost of mn , while Opt rejects $m - 1$ jobs in the beginning, incurring a cost of only $n + (m - 1)c$. This gives a competitive ratio of m as $n \rightarrow \infty$.

A complaint one might have about the job-idle time measure is that it gives the machine credit for time spent processing jobs that are later rejected. For example, if we get a job at time 0, work on it for 3 time units, and reject it at time 5, we pay $c + 2$ rather than $c + 5$. A natural alternative would be to define the cost so that no credit is given for time spent processing jobs that end up getting rejected. Unfortunately, that definition makes it impossible to achieve any finite competitive ratio. In particular, if a very large job arrives at time 0, we cannot reject it since it may be the only job and OPT would be 0; but, then if unit-size jobs appear at every time step starting at time tc , we have committed to cost tc whereas OPT could have rejected the big job at the start for a cost of only c .

The main results of this paper are as follows: In section 2, we give a 2-competitive online algorithm for flow time and job-idle time with penalty. Note that, for job-idle time, this matches the simple lower bound given above. The online algorithm is extended to an $O(\log^2 W)$ algorithm for *weighted* flow time in Section 3, where W is the ratio between the maximum and minimum weight of any job. In Section 4 we give lower bounds for the problem with arbitrary rejection penalties and also give a $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ competitive algorithm using a $(1 + \epsilon)$ speed processor in the resource augmentation model, where C is the ratio between the maximum and the minimum penalty for any job.

1.1 Related Previous Work

Flow time is a widely used criterion for measuring performance of scheduling algorithms. For the unweighted case, it has been long known [1] that the Shortest Remaining Processing Time (SRPT) policy is optimal for this problem. The weighted problem is known to be much harder. Recently Chekuri et al [2,3] gave the first non trivial semi-online algorithm for the problem that achieves a competitive ratio of $O(\log^2 P)$. Here P is the ratio of the maximum size of any job to the minimum size of any job. Bansal et al [4] give another online algorithm achieving a ratio of $O(\log W)$, and a semi-online algorithm which is $O(\log n + \log P)$ competitive. Also related is the work of Becchetti et al [5], who give a $(1 + 1/\epsilon)$ competitive algorithm for weighted flow time using a $(1 + \epsilon)$ speed processor.

Admission control has been studied for a long time in circuit routing problems (see, e.g., [6]). In these problems, the focus is typically on approximately maximizing the throughput of the network. In scheduling problems, the model of rejection with penalty was first introduced by Bartal et al [7]. They considered the problem of minimizing makespan on multiple machines with rejection and

gave a $1 + \phi$ approximation for the problem where ϕ is the golden ratio. Variants of this problem have been subsequently studied by [8,9]. Seiden [8] extends the problem to a pre-emptive model and improves the ratio obtained by [7] to 2.38.

More closely related to our work is the model considered by Engels et al [10]. They consider the problem of minimizing weighted *completion* time with rejections. However, there are some significant differences between their work and ours. First, their metric is different. Second, they only consider the offline problem and give a constant factor approximation for a special case of the problem using LP techniques.

1.2 Notation and Definitions

We consider the problem of online pre-emptive scheduling of jobs so as to minimize flow time with rejections or job idle time with rejections. Jobs arrive online; their processing time is revealed as they arrive. A problem instance $\mathcal{J}$ consists of n jobs and a penalty c . Each job j is characterized by its release time r_j and its processing time p_j . P denotes the ratio of the maximum processing time to the minimum processing time.

At any point of time an algorithm can schedule or reject any job released before that time. For a given schedule S , at any time t , a job is called active if it has not been finished or rejected yet. The completion time κ_j of a job is the time at which a job is finished or rejected.

The flow time of a job is the total time that the job spends in the system, $f_j = \kappa_j - r_j$. The flow time of a schedule S denoted by $F(S)$ is the sum of flow times of all jobs. Similarly, the job idle time of a job is the total time that the job spends in queue not being processed. This is $f_j - p_j$ if the job is never rejected, or $f_j -$ (the duration for which it was scheduled) otherwise. The job idle time of a schedule denoted by $I(S)$ is the sum of job idle times of all jobs. For a given algorithm A , let R_A be the set of jobs that were rejected and let S_A be the schedule produced. Then, the flow time with rejections of the algorithm is given by $F(S_A) + c|R_A|$. Similarly the job idle time with rejections of the algorithm is given by $I(S_A) + c|R_A|$.

We use A to denote our algorithms and the cost incurred by them. We denote the Optimal algorithm and its cost by Opt .

In the weighted problem, every job has a weight w_j associated with it. Here, the objective is to minimize weighted flow time with rejections. This is given by $\sum_j (w_j f_j) + c|R_A|$. W denotes the ratio of weights of the highest weight class and the least weight class. As in the unweighted case, the weight of a job is revealed when the job is released.

We also consider the case when different jobs have different penalties. In this case, we use c_j to denote the penalty of job j . c_{max} denotes the maximum penalty and c_{min} the minimum penalty. We use C to denote the ratio c_{max}/c_{min} .

Our algorithms do not assume knowledge of C , W or P . Finally, by a stream of jobs of size x , we mean a string of jobs each of size x , arriving every x units of time.

1.3 Preliminaries

We first consider some properties of the optimal solution (Opt) which will be useful in deriving our results.

Fact 1 *If Opt rejects a job j , it is rejected the moment it arrives.*

Fact 2 *Given the set of jobs that Opt rejects, the remaining jobs must be serviced in Shortest Remaining Processing Time (SRPT) order.*

Fact 3 *In the uniform penalty model, if a job j is rejected, then it must be the job that currently has the largest remaining time.*

2 An Online Algorithm

In this section, we will give online algorithms for minimizing flow time and job idle time with rejections.

2.1 Minimizing Flow Time

Flow time of a schedule can be expressed as the sum over all time steps of the number of jobs in the system at that time step. Let ϕ be a counter that counts the flow time accumulated until the current time step. The following algorithm achieves 2-competitiveness for flow time with rejections:

The Online Algorithm. Starting with $\phi = 0$, at every time step, increment ϕ by the number of active jobs in the system at that time step. Whenever ϕ crosses a multiple of c , reject the job with the largest remaining time. Schedule active jobs in SRPT order.

Let the schedule produced by the above algorithm be S and the set of rejected jobs be R .

Lemma 1. *The cost of the algorithm is $\leq 2\phi$.*

Proof. This follows from the behavior of the algorithm. In particular, $F(S)$ is equal to the final value in the counter ϕ , and the total rejection cost $c|R|$ is also at most ϕ because $|R|$ increases by one (a job is rejected) every time ϕ gets incremented by c .

The above lemma implies that to get a 2-approximation, we only need to show that $\phi \leq Opt$. Let us use another counter ψ to account for the cost of Opt . We will show that the cost of Opt is at least ψ and at every point of time $\psi \geq \phi$. This will prove the result.

The counter ψ works as follows: Whenever Opt rejects a job, ψ gets incremented by c . At other times, if $\phi = \psi$, then ϕ and ψ increase at the same rate (i.e. ψ stays equal to ϕ). At all other times ψ stays constant. By design, we have the following:

Fact 4 *At all points of time, $\psi \geq \phi$.*

Let $k = \lfloor \frac{\psi}{c} \rfloor - \lfloor \frac{\phi}{c} \rfloor$. Let n_o and n_a denote the number of active jobs in *Opt* and *A* respectively. Arrange and index the jobs in *Opt* and *A* in the order of decreasing remaining time. Let us call the k longest jobs of *A* *marked*. We will now prove the following:

Lemma 2. *At all times $n_o \geq n_a - k$.*

Lemma 2 will imply $Opt \geq \psi$ (and thus, 2-competitiveness) by the following argument: Whenever ψ increases by c , *Opt* spends the same cost in rejecting a job. When ψ increases at the same rate as ϕ , we have that $\psi = \phi$. In this case $k = 0$ and thus *Opt* has at least as many jobs in system as the online algorithm. Since the increase in ϕ (and thus ψ) accounts for the flow time accrued by the online algorithm, this is less than the flow time accrued by *Opt*. Thus the cost of *Opt* is bounded below by ψ and we are done.

We will prove Lemma 2 by induction over time. For this we will need to establish a suffix lemma. We will ignore the marked jobs while forming suffixes.

Let $P_o(i)$ (called a suffix) denote the sum of remaining times of jobs $i, \dots, n_o$ in *Opt*. Let $P_a(i)$ denote the sum of remaining times of jobs $i + k, \dots, n_a$ in *A* ($i, \dots, n_a - k$ among the unmarked jobs). For instance, Figure 1 below shows the suffices for $i = 2$ and $k = 2$.

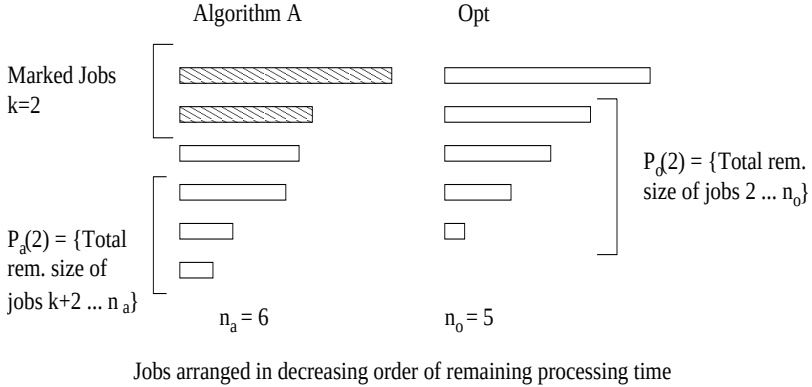


Fig. 1. Notation used in proof of Theorem 1

Lemma 3. *At all times, for all i , $P_a(i) \leq P_o(i)$.*

Proof. (of Lemma 2 using Lemma 3) Using $i = n_a - k$, we have $P_o(n_a - k) \geq P_a(n_a - k) > 0$. Therefore, $n_o \geq n_a - k$.

Proof. (Lemma 3) We prove the statement by induction over the various events in the system. Suppose the result holds at some time t . First consider the simpler

case of no arrivals. Furthermore, assume that the value of k does not change from time t to $t + 1$. Then, as A always works on the job n_a , $P_a(i)$ decreases by 1 for each $i \leq n_a - k$ and by 0 for $i > n_a - k$. Since $P_o(i)$ decreases by at most 1, the result holds for this case.

If the value of k changes between t and $t + 1$, then since there are no arrivals (by assumption), it must be the case that A rejects some job(s) and k decreases. However, note that rejection of jobs by A does not affect any suffix under A (due to the way $P_a(i)$ is defined). Thus the argument in the previous paragraph applies to this case.

We now consider the arrival of a job J at time t . If J is rejected by Opt , the suffixes of Opt remain unchanged and the value of k increases by 1. If J gets marked under A , none of the suffixes under A change either, and hence the invariant remains true. If J does not get marked, some other job with a higher remaining time than J must get marked. Thus the suffixes of A can only decrease.

If J is not rejected by Opt , we argue as follows: Consider the situation just before the arrival of J . Let C be the set of unmarked jobs under A and D the set of all jobs under Opt . On arrival of J , clearly J gets added to D . If J is unmarked under A it gets added to C else if it gets marked then a previously marked job $J' \in A$, with a smaller remaining time than J gets added to C . In either case, the result follows from Lemma 4 (see Proposition A.7, Page 120 in [11] or Page 63 in [12]), which is a result about suffixes of sorted sequences, by setting $C = C$, $D = D$, $d' = J$ and $c' = J$ or J' .

Lemma 4. *Let $C = \{c_1 \geq c_2 \geq \dots\}$ and $D = \{d_1 \geq d_2 \geq \dots\}$ be sorted sequences of non-negative numbers. We say that $C \prec D$ if $\sum_{j \geq i} c_j \leq \sum_{j \geq i} d_j$ for all $i = 1, 2, \dots$. Let $C \cup \{c'\}$ be the sorted sequence obtained inserting c' in C . Then, $C \prec D$ and $c' \leq d' \Rightarrow C \cup \{c'\} \prec D \cup \{d'\}$.*

Thus we have the following theorem:

Theorem 1. *The above online algorithm is 2-competitive with respect to Opt for the problem of minimizing flow time with rejections.*

2.2 Minimizing Job Idle Time

Firstly note that the job idle time of a schedule can be computed by adding the contribution of the jobs waiting in the queue (that is, every job except the one that is being worked upon, contributes 1) at every time step.

The same online algorithm as in the previous case works for minimizing job idle time with the small modification that the counter ϕ now increments by the number of waiting jobs at every time step. The analysis is similar and gives us the following theorem:

Theorem 2. *The above online algorithm is 2-competitive with respect to Opt for the problem of minimizing job idle time with rejections.*

2.3 Varying Server Speeds

For a researcher managing his/her to-do list, one typically has different amounts of time available on different days. We can model this as a processor whose speed changes over time in some unpredictable fashion (i.e., the online algorithm does not know what future speeds will be in advance). This type of scenario can easily fool some online algorithms: e.g., if the algorithm immediately rejected any job of size $\geq c$ according to the current speed, then this would produce an unbounded competitive ratio if the processor immediately sped up by a large factor.

However, our algorithm gives a 2-approximation for this case as well. The only effect of varying processor speed on the problem is to change sizes of jobs as time progresses. Let us look at the problem from a different angle: the job sizes stay the same, but time moves at a faster or slower pace. The only effect this has on our algorithm is to change the time points at which we update the counters ϕ and ψ . However, notice that our algorithm is locally optimal: at all points of time the counter ψ is at most the cost of Opt , and $\phi \leq \psi$, irrespective of whether the counters are updated more or less often. Thus the same result holds.

2.4 Lower Bounds

We now give a matching lower bound of 2 for waiting time and 1.5 for flow time, on the competitive ratio of any deterministic online algorithm.

Consider the following example: Two jobs of size 1 arrive at $t = 0$. The adversary gives a stream of unit size jobs starting at $t = 1$ until the algorithm rejects a job.

Let x be the time when the algorithm first rejects a job. In the waiting time model, the cost of the algorithm is $x + c$. The cost of the optimum is $\min(c, x)$, since it can either reject a job in the beginning, or not reject at all. Thus we have a competitive ratio of 2.

The same example gives a bound of 1.5 for flow time. Note that the cost of the online algorithm is $2x + c$, while that of the optimum is $\min(x + c, 2x)$.

Theorem 3. *No online algorithm can achieve a competitive ratio of less than 2 for minimizing waiting time with rejections or a competitive ratio of less than 1.5 for minimizing flow time with rejections.*

3 Weighted Flow Time with Weighted Penalties

In this section we consider the minimization of weighted flow time with admission control. We assume that each job has a weight associated with it. Without loss of generality, we can assume that the weights are powers of 2. This is because rounding up the weights to the nearest power of 2 increases the competitive ratio by at most a factor of 2. Let $a_1, a_2, \dots, a_k$ denote the different possible weights, corresponding to weight classes $1, 2, \dots, k$. Let W be the ratio of maximum to minimum weight. Then, by our assumption, k is at most $\log W$. We will consider

the following two models for penalty. The general case of arbitrary penalties is considered in the next section.

Uniform penalty: Jobs in each weight class have the same penalty c of rejection.

Proportional penalty: Jobs in weight class j have rejection penalty $a_j c$.

For both these cases, we give an $O(\log^2 W)$ competitive algorithm. This algorithm is based on the Balanced SRPT algorithm due to Bansal *et al.* [4]. We modify their algorithm to incorporate admission control. The modified algorithm is described below.

Algorithm Description: As jobs arrive online, they are classified according to their weight class. Consider the weight class that has the minimum total remaining time of jobs. Ties are resolved in favor of higher weight classes. At each time step, we pick the job in this weight class with smallest remaining time and schedule it.

Let ϕ be a counter that counts the total weighted flow time accumulated until current time step. For each weight class j , whenever ϕ crosses the penalty c (resp. $a_j c$), we reject a job with the largest remaining time from this class.

Analysis: We will imitate the analysis of the weighted flow time algorithm. First we give an upper bound on the cost incurred by the algorithm. Let $F(S)$ be the final value of counter ϕ . The cost of rejection, $c|R|$, is bounded by $k\phi$, because rejections $|R_j|$ in weight class j increase by 1 every time ϕ increases by c_j . Thus we have,

Lemma 5. *The total cost of the algorithm is $\leq (k + 1)\phi$*

In order to lower bound the cost of optimal offline algorithm, we use a counter ψ . The counter ψ works as follows: Whenever *Opt* rejects a job of weight class j , ψ gets incremented by c_j . At other times, if $\phi = \psi$, then ϕ and ψ increase at the same rate (i.e. ψ stays equal to ϕ), otherwise, ψ stays constant. By design, we have the following:

Fact 5 *At all points of time, $\psi \geq \phi$.*

Now we show that ψ is a lower bound on $k \cdot \text{Opt}$. Let $m_j = \lfloor \frac{\psi}{kc_j} \rfloor - \lfloor \frac{\phi}{kc_j} \rfloor$. In both *Opt* and our algorithm, arrange active jobs in each weight class in decreasing order of remaining processing time. We call the first m_j jobs of weight class j in our algorithm as marked. Now ignoring the marked jobs, we can use theorem 2 from Bansal *et al.* [4]. We get the following:

Lemma 6. *The total weight of unmarked jobs in our algorithm is no more than k times the total weight of jobs in *Opt*.*

Proof. (Sketch) The proof follows along the lines of lemma 2 in Bansal *et al.* [4]. Their proof works in this case if we only consider the set of unmarked jobs in our algorithm. However, due to rejections, we need to check a few more cases.

We first restate their lemma in terms suitable for our purpose. Let $B(j, l)$ and $P(j, l)$ denote a *prefix* of the jobs in our algorithm and *Opt* algorithm respectively.

Then, we define the suffixes $\overline{B}(j, l) = J_a - B(j, l)$ and $\overline{P}(j, l) = J_o - P(j, l)$, where J_a and J_o are the current sets of jobs in our algorithm and the *Opt* algorithm respectively.

Lemma 7. ([4]) *The total remaining time of the jobs in the suffix $\overline{B}(j, l)$ is smaller than the total remaining time of the jobs in $\overline{P}(j, l)$.*

We now consider the cases that are not handled by Bansal *et al.*'s proof. If a job of weight class j arrives and *Opt* rejects it, then the set of jobs with *Opt* does not change. On the other hand, m_j increases by at least 1. In our algorithm, if the new job is among top m_j jobs in its weight class, then it is marked and set of unmarked jobs remains the same. If the new job does not get marked, the suffixes of our algorithm can only decrease, since some other job with higher remaining time must get marked.

Similarly, when our algorithm rejects a job of class j , then the number of marked jobs m_j reduces by 1. However, the rejected job had highest remaining time in the class j . Hence none of the suffixes change.

Thus, we have established that the suffixes in our algorithm are smaller than the corresponding suffixes in the *Opt* algorithm at all times. The argument from Theorem 2 in [4] gives us the result that weight of unmarked jobs in our algorithm is at most $k \cdot \text{Opt}$.

To finish the argument, note that when the *Opt* algorithm rejects a job of weight class j , *Opt* increases by c_j . And ψ increases by kc_j . On the other hand, when ψ and ϕ increase together, we have $\psi = \phi$. There are no marked jobs, since $m_j = 0$ for all j . The increase in ψ per time step is same as the weight of all jobs in our algorithm. As we saw in the Lemma 6, this is at most k times the total weight of jobs in *Opt* algorithm. Thus, the total increase in ψ is bounded by $k \cdot \text{Opt}$.

In conjunction with Lemma 5, this gives us $O(\log^2 W)$ competitiveness.

4 Weighted Flow Time with Arbitrary Penalties

In this section we will consider the case when different jobs have different weights and different penalties of rejection. First we will show that even for the simpler case of minimizing unweighted flow time with two different penalties, no algorithm can obtain a competitive ratio of less than $n^{\frac{1}{4}}$ or less than $C^{\frac{1}{2}}$. A similar bound holds even if there are two different penalties and the arrival times of high penalty jobs are known in advance. Then we will give an online algorithm that achieves a competitive ratio of $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ using a processor of speed $(1 + \epsilon)$.

4.1 Lower Bounds

Theorem 4. *For the problem of minimizing flow time or job idle time with rejection, and arbitrary penalties, no (randomized) online algorithm can achieve*

a competitive ratio of less than $n^{\frac{1}{4}}$ or $C^{\frac{1}{2}}$. Even when there are only two different penalties and the algorithm has knowledge of the high penalty jobs, no online (randomized) algorithm can achieve a competitive ratio of less than $n^{\frac{1}{5}}$.

Proof. (Sketch) Consider the following scenario for a deterministic algorithm. The adversary gives two streams, each beginning at time $t = 0$. Stream1 consists of k^2 jobs, each of size 1 and penalty k^2 . Stream2 consists of k jobs each of size k and infinite penalty.

Depending on the remaining work of the online algorithm by time k^2 , the adversary decides to give a third stream of jobs, or no more jobs. Stream 3 consists of $m = k^4$ jobs, each of size 1 and infinite penalty. Let y denote the total remaining work of jobs of Stream2 that are left at time $t = k^2$. The adversary gives Stream3 if $y \geq k^2/2$.

In either case, one can show that the ratio of the optimal cost to the online cost is at $\Omega(k)$, which implies a competitive ratio of $\Omega(n^{1/4})$. Due to lack of space, the details are deferred to the full version.

Clearly, the lower bound extends to the randomized case, as the adversary can simply send Stream3 with probability $1/2$. Finally, to obtain a lower bound on competitive ratio in terms of C , we simply replace the infinite penalties of jobs in Stream2 and Stream3 by penalties of k^4 .

The bound for the case when the high penalty jobs are known is similar and deferred to the full version of the paper.

4.2 Algorithm with Resource Augmentation

Now we will give a resource augmentation result for the weighted case with arbitrary penalties. The resource augmentation model is the one introduced Kalyanasundaram and Pruhs [13], where the online algorithm is provided a $(1 + \epsilon)$ times faster processor than the optimum offline adversary.

Consider first, a fractional model where we can reject a fraction of a job. Rejecting a fraction f of job j has a penalty of fc_j . The contribution to the flow time is also fractional: If an f fraction of a job is remaining at time t , it contributes fw_j to the weighted flow time at that moment.

Given an instance of the original problem, create a new instance as follows: Replace a job j of size p_j , weight w_j and penalty c_j , with c_j jobs, each of weight w_j/c_j , size p_j/c_j and penalty 1.

Using the $O(\log^2 W)$ competitive algorithm for the case of arbitrary weights and uniform penalty, we can solve this fractional version of the original instance to within $O((\log W + \log C)^2)$. Now we use a $(1 + \epsilon)$ speed processor to convert the fractional schedule back to a schedule for the original metric without too much blowup in cost, as described below.

Denote the fractional schedule output in the first step by S_F . The algorithm works as follows: If S_F rejects more than an $\epsilon/2$ fraction of some job, reject the job completely. Else, whenever S_F works on a job, work on the same job with a $(1 + \epsilon)$ speed processor. Notice that when the faster processor finishes the job, S_F still has $1 - \epsilon/2 - 1/(1 + \epsilon) = O(\epsilon)$ fraction of the job present.

We lose at most $2/\epsilon$ times more than S_F in rejection penalties, and at most $O(1/\epsilon)$ in accounting for flow time. Thus we have the following theorem:

Theorem 5. *The above algorithm is $O(\frac{1}{\epsilon}(\log W + \log C)^2)$ -competitive for the problem of minimizing weighted flow time with arbitrary penalties on a $(1 + \epsilon)$ -speed processor.*

5 Conclusion

In this paper, we give online algorithms for the problems of minimizing flow time and job idle time when rejections are allowed at some penalty, and examine a number of problem variants. There are several problems left open by our work. It would be interesting to close the gap between the 1.5 lower bound and our 2-competitive algorithm for minimizing flow time with uniform penalties. The hardness of the offline version for the case of flow-time with uniform penalties is also not known².

References

1. Smith, W.: Various optimizers for single stage production. *Naval Research Logistics Quarterly* **3** (1956) 59–66
2. Chekuri, C., Khanna, S.: Approximation schemes for preemptive weighted flow time. *ACM Symposium on Theory of Computing (STOC)* (2002)
3. Chekuri, C., Khanna, S., Zhu, A.: Algorithms for weighted flow time. *STOC* (2001)
4. Bansal, N., Dhamdhere, K.: Minimizing weighted flow time. In: *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. (2003) 508–516
5. Becchetti, L., Leonardi, S., Spaccamela, A.M., Pruhs, K.: Online weighted flow time and deadline scheduling. In: *RANDOM-APPROX.* (2001) 36–47
6. Borodin, A., El-Yaniv, R.: *On-Line Computation and Competitive Analysis*. Cambridge University Press (1998)
7. Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., Stougie, L.: Multiprocessor scheduling with rejection. In: *ACM-SIAM Symposium on Discrete Algorithms (SODA)*. (1996)
8. Seiden, S.S.: Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science* **262** (2001) 437–458
9. Hoogeveen, H., Skutella, M., Woeginger, G.: Preemptive scheduling with rejection. *European Symposium on Algorithms* (2000)
10. Engels, D., Karger, D., Kolliopoulos, S., Sengupta, S., Uma, R., Wein, J.: Techniques for scheduling with rejection. *European Symposium on Algorithms* (1998) 490–501
11. Marshall, A.W., Olkin, I.: *Inequalities: Theory of Majorization and Its Applications*. Academic Press (1979)
12. Hardy, G., Littlewood, J.E., Polya, G.: *Inequalities*. Cambridge University Press (1952)
13. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. *JACM* **47** (2000) 617–643

² We can give a quasi-polynomial time approximation scheme (a $1 + \epsilon$ approximation with running time $n^{O(\log n/\epsilon^2)}$). This is deferred to the full version of the paper.

On Approximating a Geometric Prize-Collecting Traveling Salesman Problem with Time Windows

Extended Abstract

Reuven Bar-Yehuda¹, Guy Even², and Shimon (Moni) Shahar³

¹ Computer Science Dept., Technion, Haifa 32000, Israel. reuven@cs.technion.ac.il

² Electrical-Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel. guy@eng.tau.ac.il

³ Electrical-Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel. moni@eng.tau.ac.il

Abstract. We study a scheduling problem in which jobs have locations. For example, consider a repairman that is supposed to visit customers at their homes. Each customer is given a time window during which the repairman is allowed to arrive. The goal is to find a schedule that visits as many homes as possible. We refer to this problem as the Prize-Collecting Traveling Salesman Problem with time windows (TW-TSP).

We consider two versions of TW-TSP. In the first version, jobs are located on a line, have release times and deadlines but no processing times. A geometric interpretation of the problem is used that generalizes the Erdős-Szekeres Theorem. We present an $O(\log n)$ approximation algorithm for this case, where n denotes the number of jobs. This algorithm can be extended to deal with non-unit job profits.

The second version deals with a general case of asymmetric distances between locations. We define a density parameter that, loosely speaking, bounds the number of zig-zags between locations within a time window. We present a dynamic programming algorithm that finds a tour that visits at least $OPT/density$ locations during their time windows. This algorithm can be extended to deal with non-unit job profits and processing times.

1 Introduction

We study a scheduling problem in which jobs have locations. For example, consider a repairman that is supposed to visit customers at their homes. Each customer is given a time window during which the repairman is allowed to arrive. The goal is to find a schedule that visits as many homes as possible. We refer to this problem as the Prize-Collecting Traveling Salesman Problem with time windows (TW-TSP).

Previous Work. The goal in previous works on scheduling with locations differs from the goal we consider. The goal in previous works is to minimize the makespan (i.e. the completion time of the last job) or minimize the total waiting

time (i.e. the sum of times that elapse from the release times till jobs are served). Tsitsiklis [T92] considered the special case in which the locations are on a line. Tsitsiklis proved that verifying the feasibility of instances in which both release times and deadlines are present is strongly NP-complete. Polynomial algorithms were presented for the cases of (i) either release times or deadlines, but not both, and (ii) no processing time. Karuno et. al. [KNI98] considered a single vehicle scheduling problem which is identical to the problem studied by Tsitsiklis (i.e. locations on a line and minimum makespan). They presented a 1.5-approximation algorithm for the case without deadlines (processing and release times are allowed). Karuno and Nagamochi [KN01] considered multiple vehicles on a line. They presented a 2-approximation algorithm for the case without deadlines. Augustine and Seiden [AS02] presented a PTAS for single and multiple vehicles on trees with a constant number of leaves.

Our results. We consider two versions of TW-TSP. In the first version, TW-TSP on a line, jobs are located on a line, have release times, deadlines, but no processing times. We present an $O(\log n)$ approximation algorithm for this case, where n denotes the number of jobs. Our algorithm also handles a weighted case, in which a profit $p(v)$ is gained if location v is visited during its time window.

The second version deals with a general case of asymmetric distances between locations (asymmetric TW-TSP). We define a density parameter that, loosely speaking, bounds the number of zig-zags between locations within a time window. We present a dynamic programming algorithm that finds a tour that visits at least $OPT/density$ locations during their time windows. This algorithm can be extended to deal with non-unit profits and processing times.

Techniques. Our approach is motivated by a geometric interpretation. We reduce TW-TSP on a line to a problem called MAX-MONOTONE-TOUR. In MAX-MONOTONE-TOUR, the input consists of a collection of slanted segments in the plane, where the slope of each segment is 45 degrees. The goal is to find an x -monotone curve starting at the origin that intersects as many segments as possible. MAX-MONOTONE-TOUR generalizes the longest monotone subsequence problem [ES35]. A basic procedure in our algorithms involves the construction of an arc weighted directed acyclic graph and the computations of a max-weight path in it [F75]. Other techniques include interval trees and dynamic programming algorithms.

Organization. In Section 2, we formally define TW-TSP. In Section 3, we present approximation algorithms for TW-TSP on a line. We start with an $O(1)$ -approximation algorithm for the case of unit time-windows and end with an $O(\log n)$ -approximation algorithm. In Section 4 we present algorithms for the non-metric version of TW-TSP.

2 Problem Description

We define the Prize-Collecting Traveling Salesman Problem with time-windows (TW-TSP) as follows. Let (V, ℓ) denote a metric space, where V is a set of points and ℓ is a metric. The input of a TW-TSP instance over the metric space (V, ℓ) consists of: (i) A subset $S \subseteq V$ of points. (ii) Each element $s \in S$ is assigned a *profit* $p(s)$, a *release time* $r(s)$, and *deadline* $d(s)$. (iii) A special point $v_0 \in S$, called the *origin*, for which $p(v_0) = r(v_0) = d(v_0) = 0$.

The points model cities in TSP jargon or jobs in scheduling terminology. The distance $\ell(u, v)$ models the amount of time required to travel from u to v . We refer to the interval $[(r(v), d(v))]$ as the *time window* of v . We denote the time window of v by I_v .

A *tour* is a sequence of pairs (v_i, t_i) , where $v_i \in V$ and t_i is an arrival time. (Recall that the point v_0 is the origin.) The feasibility constraints for a tour $\{(v_i, t_i)\}_{i=0}^k$ are as follows:

$$\begin{aligned} t_0 &= 0 \\ t_{i+1} &\geq t_i + \ell(v_i, v_{i+1}). \end{aligned}$$

A *TW-tour* is a tour $\{(v_i, t_i)\}_{i=0}^k$ that satisfies the following conditions:

1. The tour is simple (multiplicity of every vertex is one).
2. For every $0 \leq i \leq k$, $v_i \in S$.
3. For every $0 \leq i \leq k$, $t_i \in I_{v_i}$.

The *profit* of a TW-tour $\mathcal{T} = \{(v_i, t_i)\}_{i=0}^k$ is defined as $p(\mathcal{T}) = \sum_{i=0}^k p(v_i)$.

The goal in TW-TSP is to find a TW-tour with maximum profit.

We refer to TW-tours simply as sequences of points in S without attaching times since we can derive feasible times that satisfy $t_i \in I_{v_i}$ as follows:

$$\begin{aligned} t_0 &= 0 \\ t_i &= \max\{t_{i-1} + \ell(v_{i-1}, v_i), r(v_i)\}. \end{aligned} \tag{1}$$

One can model multiple jobs residing in the same location (but with different time windows) by duplicating the point and setting the distance between copies of the same point to zero (hence the metric becomes a semi-metric).

3 TW-TSP on a Line

In this section we present approximation algorithms for TW-TSP on a line. TW-TSP on a line is a special case of TW-TSP in which $V = \mathbb{R}$. Namely, the points are on a the real line and $\ell(u, v) = |u - v|$.

We begin by reducing TW-TSP on a line to a geometric problem of intersecting as many slanted segments as possible using an x -monotone curve. We then present a constant ratio approximation algorithm for the special case in which the length of every time window is one. We use this algorithm to obtain

an $O(\log L)$ -approximation, where $L = \frac{\max_v |I_v|}{\min_u |I_u|}$. Finally, we present an $O(\log n)$ -approximation algorithm, where n denotes the size of S .

For simplicity we consider the case of unit point profits (i.e. $p(v) = 1$, for every v). The case of weighted profits easily follows.

3.1 A Reduction to Max-Monotone-Tour

We depict an instance of TW-TSP on a line using a two-dimensional diagram (see Fig. 1(A)). The x -axis corresponds to the value of a point. The y -axis corresponds to time. A time window $[r(v), d(v)]$ of point v is drawn as a vertical segment, the endpoints of which are: $(v, r(v))$ and $(v, d(v))$.

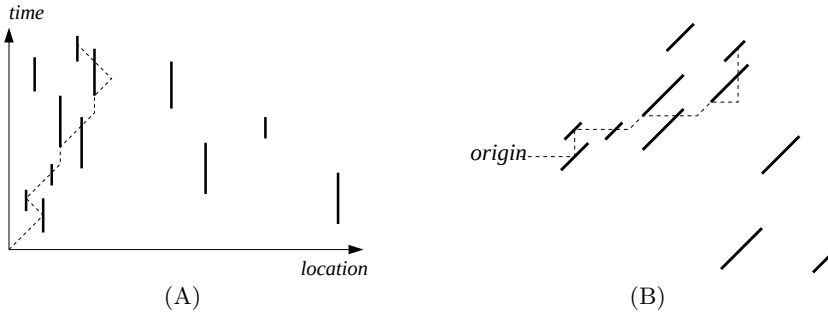


Fig. 1. (A) A two-dimensional diagram of an instance of TW-TSP on a line. (B) A MAX-MONOTONE-TOUR instance obtained after rotation by 45 degrees.

We now rotate the picture by 45 degrees. The implications are: (i) segments corresponding to time windows are segments with a 45 degree slope, and (ii) feasible tours are (weakly) x -monotone curves; namely, a curve with slopes in the range $[0, 90]$ degrees.

This interpretation reduces TW-TSP on a line to the problem of MAX-MONOTONE-TOUR defined as follows (see Fig. 1(B)). The input consists of a collection of slanted segments in the plane, where the slope of each segment is 45 degrees. The goal is to find an x -monotone curve starting at the origin that intersects as many segments as possible.

3.2 Unit Time Windows

In this section we present an 8-approximation algorithm for the case of unit time windows. In terms of the MAX-MONOTONE-TOUR problem, this means that the length of each slanted segment is 1.

We begin by overlaying a grid whose square size is $\frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}}$ on the plane. We shift the grid so that endpoints of the slanted segments do not lie on the grid lines. It follows that each slanted segment intersects exactly one vertical (resp.

horizontal) line of the grid. (A technicality that we ignore here is that we would like the origin to be a grid-vertex even though the grid is shifted). Consider a directed-acyclic graph (DAG) whose vertices are the crossings of the grid and whose edges are the vertical and horizontal segments between the vertices. We direct all the horizontal DAG edges in the positive x -direction, and we direct all the vertical DAG edges in the positive y -direction. We assign each edge e of the DAG a weight $w(e)$ that equals the number of slanted segments that intersect e . The algorithm computes a path p of maximum weight in the DAG starting from the origin. The path is the tour that the agent will use. We claim that this is an 8-approximation algorithm.

Theorem 1. *The approximation ratio of the algorithm is 8.*

We prove Theorem 1 using the two claims below. Given a path q , let $k(q)$ denote the number of slanted segments that intersect q . Let p^* denote an optimal path in the plane, and let p' denote an optimal path restricted to the grid. Let $k^* = k(p^*)$, $k' = k(p')$, and $k = k(p)$.

Claim. $k \geq k'/2$.

Proof. Let $w(q)$ denote the weight of a path q in the DAG. We claim that, for every grid-path q ,

$$w(q) \geq k(q) \geq w(q)/2.$$

The fact that $w(q) \geq k(q)$ follows directly from the definition of edge weights. The part $k(q) \geq w(q)/2$ follows from the fact that every slanted segment intersects exactly two grid edges. Hence, a slanted segment that intersects q may contribute at most 2 to $w(q)$. Since the algorithm computes a maximum weight path p , we conclude that

$$k(p) \geq w(p)/2 \geq w(p')/2 \geq k(p')/2,$$

and the claim follows.

Claim. $k' \geq k^*/4$.

Proof. Let $C_1, \dots, C_m$ denote the set of grid cells that p^* traverses. We decompose the sequence of traversed cells into blocks. Point (x_1, y_1) *dominates* point (x_2, y_2) if $x_1 \leq x_2$ and $y_1 \leq y_2$. A block B *dominates* a block B' if every point in B dominates every point in B' . Note that if point p_1 dominates point p_2 , then it is possible to travel from p_1 to p_2 along an x -monotone curve.

Let $B_1, B_2, \dots, B_{m'}$ denote the decomposition of the traversed cells into horizontal and vertical blocks. The odd indexed blocks are horizontal blocks and the even indexed blocks are vertical blocks. We present a decomposition in which B_i dominates B_{i+2} , for every i .

We define B_1 as follows. Let a_1 denote the horizontal grid line that contains the top side of C_1 . Let C_{i_1} denote the last cell whose top side is in a_1 . The block B_1 consists of the cells $C_1 \cup \dots \cup C_{i_1}$. The block B_2 is defined as follows. Let

b_2 denote the vertical grid line that contains the right side of cell C_{i_1} . Let C_{i_2} denote the last cell whose right side is in b_2 . The block B_2 consists of the cells $C_{i_1+1} \cup \dots \cup C_{i_2}$. We continue decomposing the cells into blocks in this manner. Figure 2 depicts such a decomposition.

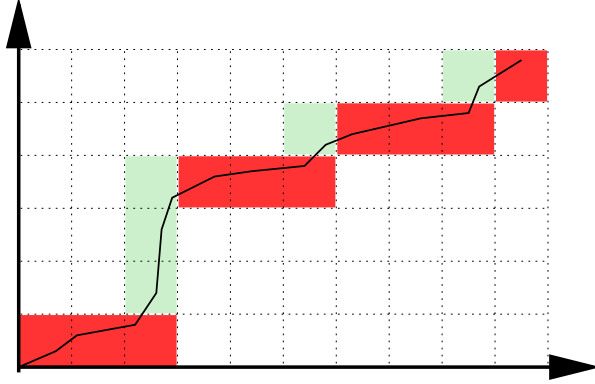


Fig. 2. A decomposition of the cells traversed by an optimal curve into alternating horizontal and vertical blocks.

Consider the first intersection of p^* with every slanted segment it intersects. All these intersection points are in the blocks. Assume that at least half of these intersection points belong to the horizontal blocks (the other case is proved analogously). We construct a grid-path $\tilde{p}$ as follows. The path $\tilde{p}$ passes through the lower left corner and upper right corner of every horizontal block. For every horizontal block, $\tilde{p}$ goes from the bottom left corner to the upper right corner along one of the following sub-paths: (a) the bottom side followed by the right side of the block, or (b) the left side followed by the top side of the block. For each horizontal block, we select the sub-path that intersects more slanted segments. The path $\tilde{p}$ hops from a horizontal block to the next horizontal block using the vertical path between the corresponding corners.

Note that if a slanted segment intersects a block, then it must intersect its perimeter at least once. This implies that, per horizontal block, $\tilde{p}$ is 2-approximate. Namely, the selected sub-path intersects at least half the slanted segments that p^* intersects in the block. Since at least half the intersection points reside in the horizontal blocks, it follows that $\tilde{p}$ intersects at least $k^*/4$ slanted segments. Since p' is an optimal path in the grid, it follows that $k(p') \geq k(\tilde{p})$, and the claim follows.

In the full version we present (i) a strongly polynomial version of the algorithm, and (ii) a reduction of the approximation ratio to $(4 + \varepsilon)$.

3.3 An $O(\log L)$ -Approximation

In this section we present an algorithm with an approximation ratio of $8 \cdot \log L$, where $L = \frac{\max_v |I_v|}{\min_u |I_u|}$. We begin by considering the case that the length of every time window is in the range $[1, 2)$.

Time windows in $[1, 2)$. The algorithm for unit time windows applies also for this case and yields the same approximation ratio. Note that the choice of grid square size and the shifting of the grid implies that each slanted segment intersects exactly one horizontal grid line and exactly one vertical grid line.

Arbitrary time windows. In this case we partition the slanted segments to length sets; the i th length set consists of all the slanted segments whose length is in the range $[2^i, 2 \cdot 2^i)$. We apply the algorithm to each length set separately, and pick the best solution. The approximation ratio of this algorithm is $8 \cdot \log L$.

In the full version we present an algorithm with a $(4 + \varepsilon) \cdot \log L$ -approximation ratio.

3.4 An $O(\log n)$ -Approximation

In this section we present an approximation algorithm for MAX-MONOTONE-TOUR with an approximation ratio of $O(\log n)$ (where n denotes the number of slanted segments). For the sake of simplicity, we first ignore the requirement that a TW-tour must start in the origin; this requirement is dealt with in the end of the section.

The algorithm is based on partitioning the set S of slanted segments to $\log n$ disjoint sets $S_1, \dots, S_{\log n}$. Each set S_i satisfies a comb-property defined as follows.

Definition 1. *A set S' of slanted segments satisfies the comb property if there exists a set of vertical lines $\mathcal{L}$ such that every segment $s \in S'$ intersects exactly one line in $\mathcal{L}$.*

We refer to a set of slanted segments that satisfy the comb property as a comb.

We begin by presenting a constant approximation algorithm for combs. We then show how a set of slanted segments can be partitioned to $\log n$ combs. The partitioning combined with the constant ratio approximation algorithm for combs yields an $O(\log n)$ -approximation algorithm.

A constant approximation algorithm for combs. Let S' denote a set of slanted segments that satisfy the comb property with respect to a set $\mathcal{L}$ of vertical lines. We construct a grid as follows: (1) The set of vertical lines is $\mathcal{L}$. (2) The set of horizontal lines is the set of horizontal lines that pass through the endpoints of slanted segments. By extending the slanted segments by infinitesimal amounts, we may assume that an optimal tour does not pass through the grid's vertices. Note that the grid consists of $2n$ horizontal lines and at most n vertical lines.

We define an edge-weighted directed acyclic graph in a similar fashion as before. The vertices are the crossings of the grid. The edges are the vertical and horizontal segments between the vertices. We direct all the horizontal DAG edges in the positive x -direction, and we direct all the vertical DAG edges in the positive y -direction. We assign each edge e of the DAG a weight $w(e)$ that equals the number of slanted segments that intersect e . The algorithm computes a maximum weight path p in the DAG. We claim that this is a 12-approximation algorithm.

Theorem 2. *The approximation ratio of the algorithm is 12.*

The proof is similar to the proof of Theorem 1 and appears in the full version.

Partitioning into combs. The partitioning is based on computing a balanced interval tree [BKOS00, p. 214]. The algorithm recursively bisects the set of intervals using vertical lines, and the comb S_i equals the set of slanted segments intersected by the bisectors belonging to the i th level. The depth of the interval tree is at most $\log n$, and hence, at most $\log n$ combs are obtained. Figure 3(A) depicts an interval tree corresponding to a set of slanted segments. Membership of a slanted segment s in a subset corresponding to a vertical line v is marked by a circle positioned at the intersection point. Figure 3(B) depicts a single comb; in this case the comb corresponding to the second level of the interval tree.

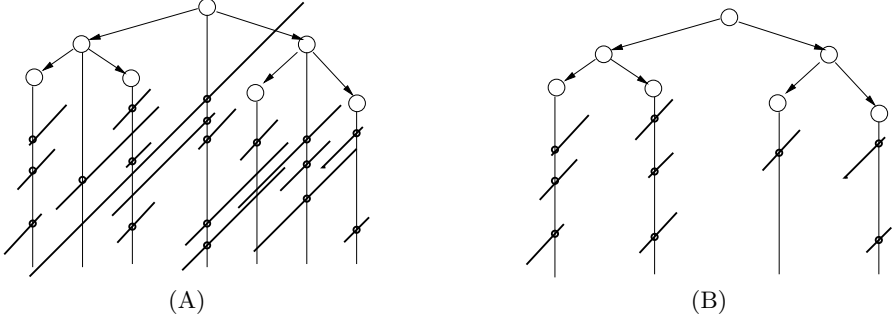


Fig. 3. (A) An interval tree corresponding to a set of slanted segments. (B) A comb induced by an interval tree.

Finding a tour starting in the origin. The approximation algorithm can be modified to find a TW-tour starting in the origin at the price of increasing the approximation ratio by a factor of two. Given a comb S_i , we consider one of two tours starting at the origin v_0 . The first tour is simply the vertical ray starting in the origin. This tour intersects all the slanted segments S'_i whose projection on the x -axis contains the origin. The second tour is obtained by running the algorithm with respect $S''_i = S_i \cup \{v_0\} \setminus S'_i$. Note that S''_i is a comb.

Remark. The algorithm for TW-TSP on a line can be easily extended to non-unit point profits $p(v)$. All one needs to do is assign grid edge e a weight $w(e)$ that equals the sum of profits of the slanted segments that intersect e . In the full version we present a $(4 + \varepsilon)$ -approximation algorithm for a comb.

4 Asymmetric TW-TSP

In this section we present algorithms for the non-metric version of TW-TSP. Asymmetric TW-TSP is a more general version of TW-TSP in which the distance function $\ell(u, v)$ is not a metric. Note that the triangle inequality can be imposed by metric completion (i.e. setting $\ell(u, v)$ to be the length of the shortest path from u to v). However, the distance function $\ell(u, v)$ may be asymmetric in this case.

4.1 Motivation

One way to try to solve TW-TSP is to (i) Identify a set of candidate arrival times for each point. (ii) Define an edge weighted DAG over pairs (v, t) , where v is a point and t is a candidate arrival times. The weight of an arc $(v, t) \rightarrow (v', t')$ equals $p(v')$. (iii) Find a longest path in the DAG with respect to edge weights.

There are two obvious obstacles that hinder such an approach. First, the number of candidate arrival times may not be polynomial. Second, a point may appear multiple times along a DAG path. Namely, a path zig-zagging back and forth to a point v erroneously counts each appearance of v as a new visit. The algorithms presented in this section cope with the problem of too many candidate points using the lexicographic order applied to sequences of arrival times of TW-tours that traverse i points (with multiplicities). The second problem is not solved. Instead we introduce a measure of density that allows us to bound the multiplicity of each point along a path.

4.2 Density of an Instance

The quality of our algorithm for asymmetric TW-TSP depends on a parameter called the *density* of an instance.

Definition 2. *The density of a TW-TSP instance Π is defined by*

$$\sigma(\Pi) = \max_{u,v} \frac{|I_u|}{\ell(u, v) + \ell(v, u)}.$$

Note that $\sigma(\Pi)$ is an upper bound on the number of “zig-zags” possible from u to v and back to u during the time window I_u . We refer to instances in which $\sigma(\Pi) < 1$ as instances that satisfy the *no-round trips within time-windows* condition.

4.3 Unit Profits and No-round Trips within Time-Windows

We first consider the case in which (i) $\sigma(\Pi) < 1$, and (ii) the profit of every point is one. In this section we prove the following theorem.

Theorem 3. *There exists a polynomial algorithm that, given an asymmetric TW-TSP instance Π with unit profits and $\sigma(\Pi) < 1$, computes an optimal TW-tour.*

Proof. Let k^* denote the maximum number of points that a TW-tour can visit. We associate with every tour $\mathcal{T} = \{v_i\}_{i=0}^{k^*}$ the sequence of arrival times $\{t_i\}_{i=0}^{k^*}$ defined in Eq. 1. Let $\mathcal{T}^* = \{(v_i^*, t_i^*)\}_{i=0}^{k^*}$ denote a TW-tour whose sequence of arrival times is lexicographically minimal among the optimal TW-tours. We present an algorithm that computes an optimal tour $\mathcal{T}$ whose sequence of arrival times equals that of $\mathcal{T}^*$.

We refer to a TW-tour of length i that ends in point v as (v, i) -lexicographically minimal if its sequence of arrival times is lexicographically minimal among all TW-tours that visit i points and end in point v . We claim that every prefix of $\mathcal{T}^*$ is also lexicographically minimal. For the sake of contradiction, consider a TW-tour $\mathcal{S} = \{u_j\}_{j=0}^i$ in which $u_i = v_i^*$ and the arrival time to u_i in $\mathcal{S}$ is less than t_i^* . We can substitute $\mathcal{S}$ for the prefix of $\mathcal{T}^*$ to obtain a lexicographically smaller optimal tour. The reason this substitution succeeds is that $\sigma(\Pi) < 1$ implies that $u_a \neq v_b^*$, for every $0 < a < i$ and $i < b \leq k^*$.

The algorithm is a dynamic programming algorithm based on the fact that every prefix of $\mathcal{T}^*$ is lexicographically minimal. The algorithm constructs layers $L_0, \dots, L_{k^*}$. Layer L_i contains a set of states (v, t) , where v denotes the endpoint of a TW-tour that arrives at v at time t . Moreover, every state (v, t) in L_i corresponds to a (v, i) -lexicographically minimal TW-tour. Layer L_0 simply contains the state $(v_0, 0)$ that starts in the origin at time 0. Layer L_{j+1} is constructed from layer L_j as described in Algorithm 1. If L_{j+1} does not contain a state with u as its point, then (u, t') is added to L_{j+1} . Otherwise, let $(u, t'') \in L_{j+1}$ denote the state in L_{j+1} that contains u as its point. The state (u, t') is added to L_{j+1} if $t' < t''$. If (u, t') is added, then (u, t'') is removed from L_{j+1} . Note that each layer contains at most n states, namely, at most one state per point. The algorithm stops as soon as the next layer L_{j+1} is empty. Let L_j denote the last non-empty layer constructed by the algorithm. The algorithm picks a state $(v, t) \in L_j$ with a minimal time and returns a TW-tour (that visits j points) corresponding to this state.

Algorithm 1 Construct layer L_{j+1}

- 1: **for all** state $(v, t) \in L_j$, and every $u \neq v$ **do**
 - 2: $t' \leftarrow \max(r(u), t + \ell(v, u))$
 - 3: **if** $t' < d(u)$ **then**
 - 4: $L_{j+1} \leftarrow \text{replace-if-min}(L_{j+1}, (u, t'))$.
 - 5: **end if**
 - 6: **end for**
-

The correctness of the algorithm is based on the following claim, the proof of which appears in the full version.

Claim. (i) If $\mathcal{T}$ is a (v, i) -lexicographically minimal TW-tour that arrives in v at time t , then $(v, t) \in L_i$; and (ii) Every state (v, t) in layer L_i corresponds to a (v, i) -lexicographically minimal TW-tour.

Part (ii) of the previous claim implies that the last layer constructed by the algorithm is indeed L_{k^*} . Since every prefix of $\mathcal{T}^*$ is lexicographically minimal, it follows that layer L_i contains the state (v_i^*, t_i^*) . Hence, the algorithm returns an optimal TW-tour. This TW-tour also happens to be lexicographically minimal, and the theorem follows.

4.4 Arbitrary Density

In this section we consider instances with arbitrary density and unit profits. The dynamic programming algorithm in this case proceeds as before but may construct more than k^* layers. We show that at most $k^* \cdot (\lfloor \sigma(\Pi) \rfloor + 1)$ layers are constructed. A path q corresponding to a state in layer L_j may not be simple, and hence, $k(q)$ (the actual number of visited points) may be less than j (the index of the layer).

Claim. The approx-ratio of the dynamic programming algorithm is $\lfloor \sigma(\Pi) \rfloor + 1$.

Proof. Consider a path $p = \{v_i\}_{i=0}^j$ corresponding to a state in layer L_j . Let t_i denote the arrival time to v_i in p . We claim that the multiplicity of every point along p is at most $\lfloor \sigma(\Pi) \rfloor + 1$. Pick a vertex v , and let $i_1 < i_2 < \dots < i_a$ denote the indexes of the appearances of v along p . Since self-loops are not allowed, it follows that between every two appearances of v , the path visits another vertex. Density implies that, for every $b = 1, \dots, a - 1$,

$$t_{i_{b+1}} - t_{i_b} \geq \frac{|I_v|}{\sigma(\Pi)}.$$

It follows that

$$t_{i_a} - t_{i_1} \geq (a - 1) \cdot \frac{|I_v|}{\sigma(\Pi)}.$$

Since $r(v) \leq t_{i_1} < t_{i_a} \leq d(v)$, it follows that $\sigma(\Pi) \geq a - 1$. We conclude that the multiplicity of v in p is at most $\lfloor \sigma(\Pi) \rfloor + 1$.

The index of the last layer found by the algorithm is at least k^* , and hence, the path computed by the algorithm visits at least $k^* / (\lfloor \sigma(\Pi) \rfloor + 1)$ points, and the claim follows.

4.5 Non-unit Profits

In the full version we consider instances of asymmetric TW-TSP with non-unit profits $p(v)$. We present (i) a trivial reduction of Knapsack to asymmetric TW-TSP and (ii) discuss a variation of the dynamic programming algorithm with an approximation ratio of $(1 + \varepsilon) \cdot (\lfloor \sigma(\Pi) \rfloor + 1)$.

4.6 Processing Times

Our algorithms for asymmetric TW-TSP can be modified to handle also processing times. The processing time of point v is denoted by $h(v)$ and signifies the amount of time that the agent must spend at a visited point. The definition of arrival times is modified to:

$$\begin{aligned} t_0 &= 0 \\ t_i &= \max\{t_{i-1} + h(v_{i-1}) + \ell(v_{i-1}, v_i), r(v_i)\}. \end{aligned} \quad (2)$$

The definition of density with processing times becomes

$$\sigma(\Pi) = \max_{u,v} \frac{|I_u|}{\ell(u, v) + \ell(v, u) + h(u) + h(v)}.$$

States are generated by the dynamic programming algorithm as follows. The arrival time t' of state (u, t') generated by state (v, t) is

$$t' = \max\{t + h(v) + \ell(v, u), r(u)\}.$$

Acknowledgments. We would like to thank Sanjeev Khanna and Piotr Krysta for helpful discussions. We especially thank Piotr for telling us about references [T92] and [KNI98]; this enabled finding all the other related references. We thank Wolfgang Slany for suggesting a nurse scheduling problem which motivated this work.

References

- [AS02] John Augustine and Steven S. Seiden, "Linear Time Approximation Schemes for Vehicle Scheduling", SWAT 2002, LNCS 2368, pp. 30–39, 2002.
- [BKOS00] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwartzkopf, "Computational Geometry – Algorithms and Applications", Springer Verlag, 2000.
- [ES35] P. Erdős and G. Szekeres, "A combinatorial problem in geometry", *Compositio Math.*, 2, 463–470, 1935.
- [F75] M. L. Fredman, "On computing the length of longest increasing subsequences", *Discrete Math.* 11 (1975), 29–35.
- [KNI98] Y. Karuno, H. Nagamochi, and T. Ibaraki, "A 1.5-approximation for single-vehicle scheduling problem on a line with release and handling times", Technical Report 98007, 1998.
- [KN01] Yoshiyuki Karuno and Hiroshi Nagamochi, "A 2-Approximation Algorithm for the Multi-vehicle Scheduling Problem on a Path with Release and Handling Times", ESA 2001, LNCS 2161, p. 218–229, 2001.
- [T92] John N. Tsitsiklis, "Special Cases of Traveling Salesman and Repairman Problems with Time Windows", *Networks*, Vol. 22, pp. 263–282, 1992.

Semi-clairvoyant Scheduling

Luca Becchetti^{1*}, Stefano Leonardi^{1 *}, Alberto Marchetti-Spaccamela^{1 *}, and Kirk Pruhs^{2**}

¹ Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”, {becchetti,leon,alberto}@dis.uniroma1.it.

² Computer Science Department
University of Pittsburgh, kirk@cs.pitt.edu

Abstract. We continue the investigation initiated in [2] of the quality of service (QoS) that is achievable by *semi-clairvoyant* online scheduling algorithms, which are algorithms that only require approximate knowledge of the initial processing time of each job, on a single machine. In [2] it is shown that the obvious semi-clairvoyant generalization of the Shortest Processing Time is $O(1)$ -competitive with respect to average stretch on a single machine. In [2] it was left as an open question whether it was possible for a semi-clairvoyant algorithm to be $O(1)$ -competitive with respect to average flow time on one single machine. Here we settle this open question by giving a semi-clairvoyant algorithm that is $O(1)$ -competitive with respect to average flow time on one single machine. We also show a semi-clairvoyant algorithm on parallel machines that achieves up to constant factors the best known competitive ratio for clairvoyant on-line algorithms. In some sense one might conclude from this that the QoS achievable by semi-clairvoyant algorithms is competitive with clairvoyant algorithms.

It is known that the clairvoyant algorithm SRPT is optimal with respect to average flow time and is 2-competitive with respect to average stretch. Thus it is possible for a clairvoyant algorithm to be simultaneously competitive in both average flow time and average stretch. In contrast we show that no semi-clairvoyant algorithm can be simultaneously $O(1)$ -competitive with respect to average stretch and $O(1)$ -competitive with respect to average flow time. Thus in this sense one might conclude that the QoS achievable by semi-clairvoyant algorithms is not competitive with clairvoyant algorithms.

1 Introduction

As observed in [2], rounding of processing times is a common/effective algorithmic technique to reduce the space of schedules of interest, thus allowing efficiently

* Partially supported by the IST Programme of the EU under contract ALCOM-FT, APPOL II, and by the MIUR Projects “Società dell’informazione”, “Algorithms for Large Data Sets: Science and Engineering” and “Efficient algorithms for sequencing and resource allocations in wireless networks”

** Supported in part by NSF grant CCR-0098752, NSF grant ANIR-0123705, and a grant from the US Air Force.

computable construction of desirable schedules. This motivated the authors of [2] to initiate a study of the quality of service (QoS) that is achievable by *semi-clairvoyant* online scheduling algorithms, which are algorithms that only require approximate knowledge of the initial processing time of each job, on a single machine. In contrast, a *clairvoyant* online algorithm requires exact knowledge of the processing time of each job, while a *nonclairvoyant* algorithm has no knowledge of the processing time of each job. An explicit categorization of what QoS is achievable by semi-clairvoyant algorithms will be useful information for future algorithms designers who wish to use rounding as part of their algorithm design.

We would also like to point out that there are applications where semi-clairvoyance arises in practice. Take for one example web servers. Currently almost all web servers use FIFO scheduling instead of SRPT scheduling even though SRPT is known to minimize average flow time, by far the most commonly accepted QoS measure, and SRPT is known to be 2-competitive with respect to average stretch [12]. The mostly commonly stated reason that web servers use FIFO scheduling is the fear of starvation. In [5] a strong argument is made that this fear of starvation is unfounded (as long as the distribution of processing times is heavily tailed, as it is the case in practice), and that web servers should adopt SRPT scheduling. Within the context of a web server, an SRPT scheduler would service the document request where the size of the untransmitted portion of the document was minimum. However document size is only an approximation of the time required by the server to handle a request, it also depends on other variables such as where the document currently resides in the memory hierarchy of the server, connection time over the Internet and associated delays etc. Furthermore, currently something like a 1/3 of web traffic consists of dynamic documents (for example popular sites such as msnbc.com personalize their homepage so that they send each user a document with their local news and weather). While a web server is constructing the dynamic content, the web server will generally only have approximate knowledge of the size of the final document.

In [2] it is shown that the obvious semi-clairvoyant generalization of the Shortest Processing Time is $O(1)$ -competitive with respect to average stretch. Surprisingly, it is shown in [2] that the obvious semi-clairvoyant generalization of SRPT, always running the job where the estimated remaining processing time is minimum, is not $O(1)$ -competitive with respect to average flow time. This result holds even if the scheduler is continuously updated with good estimates of the remaining processing time of each job. [2] gives an alternative algorithm that is $O(1)$ -competitive with respect to average flow time in this stronger model where the scheduling algorithm is continuously updated with good estimates of the remaining processing time of each job. The obvious question left open in [2] was whether there exists an $O(1)$ -competitive semi-clairvoyant algorithm for average flow time. In [2] it is conjectured that such an algorithm exists and some hints are given as to its construction.

In a personal communication, the authors of [2] proposed an algorithm, which we call R . In section 4, we show that in fact R is $O(1)$ -competitive with respect to

average flow time, thus settling positively the conjecture in [2]. In some sense one might conclude from these results that the QoS achievable by semi-clairvoyant algorithms is competitive with clairvoyant algorithms. We would like to remark that the approach we propose provides constant approximation when the processing times of jobs are known up to a constant factor and it does not use the remaining processing times of jobs.

For minimizing the average flow time on parallel machines we cannot hope for better than a logarithmic competitive ratio as there exist $\Omega(\log P)$ and $\Omega(\log n/m)$ lower bounds on the competitive ratio for clairvoyant algorithms [10]. Asymptotically tight bounds are achieved by the Shortest Remaining Processing Time heuristic (SRPT) [10]. In section 5, we show that a semi-clairvoyant greedy algorithm, which always runs a job from the densest class of jobs, can also achieve these logarithmic competitive ratios.

We now turn our attention back to the single machine case. As mentioned previously, it is known that the clairvoyant algorithm SRPT is optimal with respect to average flow time and is 2-competitive with respect to average stretch. Thus it is possible for a clairvoyant algorithm to be simultaneously competitive in both average flow time and average stretch. In contrast we show in section 6 that no semi-clairvoyant algorithm can be simultaneously $O(1)$ -competitive with respect to average stretch and $O(1)$ -competitive with respect to average flow time. Thus in this sense one might conclude that the QoS achievable by semi-clairvoyant algorithms is not competitive with clairvoyant algorithms. It is known that it is not possible for a nonclairvoyant algorithm to be $O(1)$ -competitive with respect to average flow time [11], although nonclairvoyant algorithms can be log competitive [8,3], and can be $O(1)$ -speed $O(1)$ -competitive [4,6,7].

2 Preliminaries

An instance consists of n jobs $J_1, \dots, J_n$, where job J_i has a non-negative integer release time r_i , and a positive integer processing time or length p_i . An online scheduler is not aware of J_i until time r_i . We assume that at time r_i a semi-clairvoyant algorithm learns the class c_i of job J_i , where job J_i is in class k if $p_i \in [2^k, 2^{k+1})$. Each job J_i must be scheduled for p_i time units after time r_i . Preemption is allowed, that is, the schedule may suspend the execution of a job and later begin processing that job from the point of suspension, on the same or on a different machine. The completion time C_i^S of a job J_i in the schedule S is the earliest time that J_i has been processed for p_i time units. The flow time F_i^S of a job J_i in a schedule S is $C_i^S - r_i$, and the average flow time is $\frac{1}{n} \sum_{i=1}^n F_i^S$. The stretch of a job J_i in a schedule S is $(C_i^S - r_i)/p_i$, and the average stretch is $\frac{1}{n} \sum_{i=1}^n (C_i^S - r_i)/p_i$.

A job is alive at time t if released by time t but not yet completed by the online scheduler. Alive jobs are distinguished between *partial* and *total* jobs. Partial jobs have already been executed in the past by the online scheduler, while total jobs have never been executed by the online scheduler. Denote by $\delta^A(t)$, $\rho^A(t)$, $\tau^A(t)$ respectively the number of jobs uncompleted in the algorithm

A at time t , the number of partial jobs uncompleted in the algorithm A at time t , the number of total jobs uncompleted by the algorithm A at time t . Denote by $V^A(t)$ the remaining volume, or unfinished work, for algorithm A at time t . Subscripting a variable by a restriction on the class k , restricts the variable to only jobs in classes satisfying this restriction. So for example, $V_{\leq k, > h}^A(t)$ is the remaining volume for algorithm A at time t on jobs in classes in the range $(h, k]$.

The following lemma concerning the floor function will be used in the proof.

Lemma 1. *For all x and y , $\lfloor x + y \rfloor \leq \lceil x \rceil + \lfloor y \rfloor$ and $\lfloor x - y \rfloor \leq \lfloor x \rfloor - \lfloor y \rfloor$.*

Proof. If either x or y is an integer, then the first inequality obviously holds. Otherwise, $\lfloor x + y \rfloor \leq \lfloor x \rfloor + \lfloor y \rfloor + 1 = \lceil x \rceil + \lfloor y \rfloor$, since by hypothesis both x and y are not integers.

If either x or y is an integer, then the second inequality obviously holds. Otherwise, denoted by $\{x\}$ and $\{y\}$ respectively the fractional parts of x and y , $\lfloor x - y \rfloor = \lfloor x \rfloor - \lfloor y \rfloor$ if $\{x\} - \{y\} \geq 0$, while $\lfloor x - y \rfloor = \lfloor x \rfloor - \lfloor y \rfloor - 1$ if $\{x\} - \{y\} < 0$.

3 Description of Algorithm R

The following strategy is used at each time t to decide which job to run. Note that it is easy to see that this strategy will guarantee that at all times each class of jobs has at most one partial job, and we will use this fact in our algorithm analysis.

- If at time t , all of the alive jobs are of the same class k , then run the partial job in class k if one exists, and otherwise run a total job in class k .
- Now consider the case that there are more than two classes with active jobs. Consider the two smallest integers $h < k$ such that these classes have active jobs.
 1. If h contains exactly one total job J_i and k contains exactly one partial job J_j , then run J_j . We say that the *special rule* was applied to class k at this time.
 2. In all other cases run the partial job in class h if one exists, otherwise run a total job in class h .

Observe that it is never the case that a class contains more than one partial job.

4 Analysis of Algorithm R

Our goal in this section is to prove that R is $O(1)$ -competitive with respect to flow time.

Lemma 2. *For all schedules A , $\sum_{i=1}^n F_i^A = \int_t \delta^A(t) dt$*

Lemma 2 shows that in order to prove that R is $O(1)$ -competitive with respect to flow time, it is sufficient to prove that at any time t , $\delta^{\text{OPT}}(t) = \Omega(\delta^R(t))$. We thus fix an arbitrary time t for the rest of the section, and provide a proof that this relation holds at time t . We sometimes omit t from the notation when it should be clear from the context. Any variable that doesn't specify a time, is referring to time t .

We now give a roadmap of the competitiveness proof of R ; the proof consists of three main steps:

1. We first show (Lemma 4) that in order to prove that $\delta^{\text{OPT}} = \Omega(\delta^R)$, it is sufficient to prove that $\delta^{\text{OPT}} = \Omega(\tau^R)$.
2. We then bound τ^R by showing that

$$\tau^R \leq 2\delta^{\text{OPT}} + \sum_{k_m}^{k_M-1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$$

3. We complete the proof by proving that $\sum_{k_m}^{k_M-1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$ is at most $2\delta^{\text{OPT}}$.

The proof of the bound of step 1 above hinges on the following Lemma.

Lemma 3. *If at some time t it is the case that R has partial jobs in classes h and k , with $h < k$, then R has a total job in some class in the range $[h, k]$.*

Proof. Let J_i be the partial job in h and J_j be the partial job in k . It must be the case that R first ran J_j before J_i , otherwise the fact that J_i was partial would have blocked R from later starting J_j . Consider the time s that R first started J_i . If there is a total job in a class in the range $[h, k]$ at time s , then this job will still be total at time t . If there are only partial jobs, then we get a contradiction since R would have applied the special rule at time s and would have not started J_i .

Lemma 4. $\tau^R \geq (\rho^R - 1)/2$.

Proof. Let $u_c, u_{c-1}, \dots, u_1$ be the classes with partial jobs. We consider $\lfloor c/2 \rfloor$ disjoint intervals, $[u_c, u_{c-1}]$, $[u_{c-2}, u_{c-3}]$, $\dots$. Lemma 4 implies that there is a total job in each interval. Since the intervals are disjoint these total jobs are distinct and the lemma then follows since $\lfloor c/2 \rfloor \geq (c-1)/2$ for integer c .

Before proceeding with the second step of the proof, we need a few definitions and a lemma. Let $\Delta V(s)$ be $V^R(s) - V^{\text{OPT}}(s)$. Let k_m and k_M be the minimum and maximum non-empty class at time t in R 's schedule. Let b_k be the last time, prior to t , when algorithm R scheduled a job of class higher than k . Let b_k^- be the time instant just before the events of time b_k happened.

Lemma 5. *For all classes k , $\Delta V_{\leq k}(t) < 2^{k+1}$.*

Proof. If $b_k = 0$ then obviously $\Delta V_{\leq k}(t) \leq 0$. So assume $b_k > 0$. The algorithm R has only worked on jobs of class $\leq k$ in the interval $[b_k, t)$. Hence $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(b_k)$. Further $\Delta V_{\leq k}(b_k) < 2^{k+1}$, since at time b_k^- at most one job of class $\leq k$ was in the system. A job in class $\leq k$ can be in the system at time b_k^- in case that the special rule was invoked on a job of class $> k$ at this time.

We are now ready to return to showing that $\delta^{\text{OPT}} = \Omega(\tau^R)$.

$$\begin{aligned}
\tau^R &= \sum_{k_m}^{k_M} \tau_k^R \\
&\leq \sum_{k_m}^{k_M} \lfloor \frac{V_k}{2^k} \rfloor \\
&= \sum_{k_m}^{k_M} \lfloor \frac{V_k^{\text{OPT}} + \Delta V_k}{2^k} \rfloor \\
&\leq \sum_{k_m}^{k_M} \lceil \frac{V_k^{\text{OPT}}}{2^k} \rceil + \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k} - \Delta V_{\leq k-1}}{2^k} \rfloor \\
&\leq 2\delta_{\geq k_m, \leq k_M}^{\text{OPT}} + \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k} - \Delta V_{\leq k-1}}{2^k} \rfloor \\
&\leq 2\delta_{\geq k_m, \leq k_M}^{\text{OPT}} + \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \sum_{k_m}^{k_M} \lfloor \frac{\Delta V_{\leq k-1}}{2^k} \rfloor \\
&= 2\delta_{\geq k_m, \leq k_M}^{\text{OPT}} + \lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor + \sum_{k_m}^{k_M-1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right) - \lfloor \frac{\Delta V_{\leq k_m-1}}{2^{k_m}} \rfloor
\end{aligned}$$

The fourth and the sixth line follow from the first and the second inequality of lemma 1, respectively.

Now observe that $\lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor \leq \delta_{> k_M}^{\text{OPT}}$. In fact lemma 5 implies that $\lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor \leq 1$; moreover observe that if $\Delta V_{\leq k_M} > 0$ then $\Delta V_{> k_M} < 0$ and, hence, $\delta_{> k_M}^{\text{OPT}} \geq 1$. Also note that

$$-\lfloor \frac{\Delta V_{\leq k_m-1}}{2^{k_m}} \rfloor = -\lfloor \frac{V_{\leq k_m-1}^R - V_{\leq k_m-1}^{\text{OPT}}}{2^{k_m}} \rfloor \leq \lceil \frac{V_{\leq k_m-1}^{\text{OPT}}}{2^{k_m}} \rceil \leq \delta_{\leq k_m-1}^{\text{OPT}}$$

where the first inequality follows since $V_{\leq k_m-1}^R \geq 0$ and the last inequality follows since, for each k , we have $\delta_k^{\text{OPT}} \geq \lceil \frac{V_k^{\text{OPT}}}{2^{k+1}} \rceil$.

It follows that

$$\tau^R \leq 2\delta_{\geq k_m, \leq k_M}^{\text{OPT}} + \lfloor \frac{\Delta V_{\leq k_M}}{2^{k_M}} \rfloor + \sum_{k_m}^{k_M-1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right) - \lfloor \frac{\Delta V_{\leq k_m-1}}{2^{k_m}} \rfloor$$

$$\begin{aligned}
&\leq 2\delta_{\geq k_m, \leq k_M}^{\text{OPT}} + \delta_{> k_M}^{\text{OPT}} + \sum_{k_m}^{k_M-1} \left(\left\lfloor \frac{\Delta V_{\leq k}}{2^k} \right\rfloor - \left\lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \right\rfloor \right) + \delta_{\leq k_m-1}^{\text{OPT}} \\
&\leq 2\delta^{\text{OPT}} + \sum_{k_m}^{k_M-1} \left(\left\lfloor \frac{\Delta V_{\leq k}}{2^k} \right\rfloor - \left\lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \right\rfloor \right)
\end{aligned}$$

Our final goal will be to show that $\sum_{k_m}^{k_M-1} \left(\left\lfloor \frac{\Delta V_{\leq k}}{2^k} \right\rfloor - \left\lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \right\rfloor \right)$ is at most $2\delta^{\text{OPT}}$. From this it will follow that, $\tau^R \leq 4\delta^{\text{OPT}}$. We say that R is *far behind* on a class k if $\Delta V_{\leq k}(t) \geq 2^k$. Notice that for any class k on which R is not far behind, the term $\left(\left\lfloor \frac{\Delta V_{\leq k}}{2^k} \right\rfloor - \left\lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \right\rfloor \right)$ is not positive. If it happened to be the case that R was not far behind on any class, then we would essentially be done. We thus now turn to characterizing those classes where R can be far behind. In order to accomplish this, we need to introduce some definitions.

Definition 1. For a class k and a time t , define by $s_k(t)$ the last time before time t when the special rule has been applied to class k . If $s_k(t)$ exists and R has only executed jobs of class $< k$ after time $s_k(t)$, then k is a special class at time t .

By the definition of special class it follows:

Lemma 6. If class k is special at time t then the special rule was never applied to a class $\geq k$ in $(s_k(t), t]$.

Let $u_1 < \dots < u_a$ be the special classes in R at time t . Let f_i and s_i be the first and last times, respectively, that the special rule was applied to an uncompleted job of class u_i in R . Let l_i be the unique class $< u_i$ that contains a (total) job at time s_i . We say that l_i is *associated* to u_i . Note that at time s_i , R contains a unique job in l_i , and that this job is total at this time. A special class u_i is *pure* if $l_{i+1} > u_i$, and is *hybrid* if $l_{i+1} \leq u_i$. The largest special class is by definition pure. Lemma 7 states that the special rule applications in R occur in strictly decreasing order of class. Lemma 8 states that R can not be far behind on pure classes.

Lemma 7. For all i , $1 \leq i \leq a-1$, $s_{i+1} < f_i$.

Proof. At any time $t \in [f_i, s_i]$ the schedule contains a partial job in class u_i and a total job in a class $l_i < u_i$. This implies that the special rule cannot be applied at any time $t \in [f_i, s_i]$ to a class $u_{i+1} > u_i$. Moreover, after time s_i , only jobs of class $< u_i$ are executed.

Lemma 8. For any pure special class u_k , $\Delta V_{\leq u_k}(t) \leq 0$.

Proof. If $b_{u_k} = 0$ then the statement is obvious, so assume that $b_{u_k} > 0$. Notice that no job of class $\leq u_k$ was in the system at time $b_{u_k}^-$. Otherwise, it would have to be the case that $b_k = s_{k+1}$ and $u_k \geq l_{k+1}$, which contradicts the fact that u_k is pure. We can then conclude that $\Delta V_{\leq u_k}(t) \leq \Delta V_{\leq u_k}(b_{u_k}) \leq 0$.

We now show that R can not be far behind on special classes where it has no partial job at time t .

Lemma 9. *If the schedule R is far behind on some class k falling in a maximal interval $[u_b < \dots < u_c]$, where u_c is pure, and where $[u_b < \dots < u_{c-1}]$ are hybrid, then one of the following cases must hold:*

1. $k = u_i$, where $b \leq i \leq c - 1$, where $l_{i+1} = u_i$, and R has a partial job in u_i at time t , or
2. $k = l_b$.

Proof. It is enough to show that if $k \neq l_b$ then 1 has to hold.

First note that since R is far behind on class k , it must be the case that $b_k > 0$. If R had no alive jobs of class $\leq k$ at time b_k^- then $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(b_k) \leq 0$. Since this is not the case, R was running a job in a special class u_i at time b_k , and $l_i \leq k$. By the definition of b_k it must be the case that $b_k = s_i$. If $l_i < k$ then $\Delta V_{\leq k}(t) \leq \Delta V_{\leq k}(b_k) < 2^{l_i+1} \leq 2^k$. Therefore R would not be far behind on class k . Thus we must accept the remaining alternative that $k = l_i$.

However, after s_i , R only executed jobs of class at most l_i . This, together with $f_{i-1} > s_i$, implies $u_{i-1} \leq l_i$. Furthermore, $u_{i-1} \geq l_i$ since u_{i-1} is hybrid. Thus, $u_{i-1} = l_i$.

We also need the following property.

Lemma 10. *If R is far behind on a hybrid class u_i then it must have a partial job of class u_i at time t .*

Proof. Assume by contradiction that R is far behind on class u_i but it has no partial job of class u_i at t . It must be the case that R completed a job at time s_i . At s_i there were exactly one total job of class l_i and one partial job of class u_i and no jobs in classes between l_i and u_i . Hence $\Delta V_{u_i}(s_i) \leq 0$ and $\Delta V_{\leq u_i}(s_i) = \Delta V_{\leq l_i}(s_i)$. Since by the definition of R , R didn't run a job of class $\geq u_i$ after time s_i , it must be the case that $\Delta V_{\leq u_i}(t) \leq \Delta V_{\leq u_i}(s_i) = \Delta V_{\leq l_i}(s_i) < 2^{l_i+1} \leq 2^{u_i}$.

We now essentially analyze R separately for each of the maximal subsequences defined above. Lemma 11 establishes that in the cases where $k = u_i$, OPT has an unfinished job in between any two such classes. Hence, lemma 11 associates with class u_i on which R is far behind, a unique job that OPT has unfinished at time t . Lemma 12 handles cases where $k = l_b$ by observing that OPT has at least one unfinished job in classes in the range $[l_b + 1, u_c]$. Hence, lemma 12 associates with each class l_b on which R is far behind, a unique job that OPT has unfinished at time t . From this we can conclude that the number of classes on which R is far behind, and hence the value of the term $\sum_{k_m}^{k_M-1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$ is at most $2\delta^{\text{OPT}}(t)$.

Lemma 11. *Consider a hybrid class u_i , $b \leq i < c$, on which R is far behind. Let u_j be the smallest special class larger than u_i containing a partial job at time t , for some $j \leq c$. If no such class exists then let $u_j = u_c$. Then at time t , OPT has at least one unfinished job in classes in the range $[u_i + 1, u_j]$.*

Proof. By Lemma 10, R has a partial job unfinished in u_i at time t . Now, two cases are possible.

1. If $u_j = u_c$, then by Lemma 8 we have $\Delta V_{\leq u_c}(t) \leq 0$. On the other hand, $\Delta V_{\leq u_i}(t) \geq 2^{u_i}$, since R is far behind on u_i . Hence, it has to be the case that $\Delta V_{>u_i, \leq u_c} \leq -2^{u_i}$, that is, OPT has at least one job in the interval $[u_i + 1, u_c]$.
2. Assume $u_j < u_c$. The partial job of class u_j is present over the whole interval $[s_j, t]$. This, together with Lemma 3, implies that R has a total job in a class in the range $[u_i + 1, u_j]$ at time t .
Now, let $k > u_i$ be the smallest class for which R contains a total job J at time t . Assume by contradiction that OPT does not have any unfinished job in $[u_i + 1, k]$. Hence, $\Delta V_{\leq k, \geq u_i+1} \geq 2^k$ and, since R is far behind on u_i , it is also far behind on k .
As a consequence of this fact, if $k \in [u_i + 1, u_j - 1]$ we reach a contradiction, since by Lemmas 9 and 10, it has to be the case that k is hybrid and has a partial job in k , against the hypothesis that u_j is the first such class following u_i .

Now, if $k = u_j$, then J is of class u_j and it was released after time s_j , by definition of s_j . So, R only worked on jobs of class strictly less than u_j in $(s_j, t]$, while OPT completed J in the same interval. This and Lemma 5 imply $\Delta V_{<u_j}(t) \leq \Delta V_{<u_j}(s_j) - 2^{u_j} < 0$.

If $u_i + 1 < u_j$, $\Delta V_{<u_j}(t) = \Delta V_{\leq u_i}(t) + \Delta V_{>u_i, <u_j}(t) < 0$ and $\Delta V_{\leq u_i}(t) \geq 2^{u_i}$ imply that OPT has at least one job in classes belonging to interval $[u_i + 1, u_j - 1]$ at t . Instead, if $u_i + 1 = u_j$, we have a contradiction, since R is far behind on u_i .

Lemma 12. *If $\Delta V_{\leq l_b}(t) \geq 2^{l_b}$, then at time t , OPT has at least one unfinished job in some class in the range $[l_b + 1, u_c]$.*

Proof. From $\Delta V_{\leq u_c}(t) \leq 0$ (Lemma 8) and $\Delta V_{\leq l_b}(t) \geq 2^{l_b}$, it follows $\Delta V_{>l_b, \leq u_c}(t) < 0$. That is, R is ahead of OPT on jobs in classes $(l_b, u_c]$ at time t . This means that at time t , OPT has an unfinished job in classes $(l_b, u_c]$.

We now know that the value of the term $\sum_{k_m}^{k_M-1} \left(\lfloor \frac{\Delta V_{\leq k}}{2^k} \rfloor - \lfloor \frac{\Delta V_{\leq k}}{2^{k+1}} \rfloor \right)$ is at

most $2\delta^{\text{OPT}}$. Hence, $\tau^R \leq 4\delta^{\text{OPT}}$. By applying lemma 4, which states that $\tau^R \geq (\rho^R - 1)/2$, we can conclude that $\rho^R \leq 8\delta^{\text{OPT}} + 1$. Then $\delta^R = \tau^R + \rho^R \leq 12\delta^{\text{OPT}} + 1$. Hence, we get the desired theorem.

Theorem 1. *Algorithm R is at most 13-competitive for the problem of minimizing the average flow time.*

5 Semi-clairvoyant Scheduling on Parallel Machines

In this section we present an algorithm that achieves tight logarithmic bounds even in the semi-clairvoyant case. The class of a job is defined as in the single machine case, i.e. a job is of class j if its processing time is in the range

$[2^j, 2^{j+1})$. Algorithm Lowest Class First (LCF) gives precedence to jobs of lower class, always breaking ties in favor of partial jobs. The proof of the following two theorems follow almost same lines of the proof of competitiveness of SRPT presented in [9].

Theorem 2. *Algorithm LCF is $O(\log P)$ -competitive.*

Theorem 3. *Algorithm LCF is $O(\log n/m)$ -competitive.*

6 Simultaneous Competitiveness

Theorem 4. *No semi-clairvoyant algorithm A can be both $O(1)$ -competitive with respect to average stretch and $O(1)$ -competitive with respect to average flow time.*

Proof. Assume that A is s -competitive with respect to average stretch. The lower bound construction is divided into stages. Stage i starts at time b_i . During stage i one job J_i of class c_i is released. Stage 0 starts at time $b_0 = 0$ when a job J_0 of class m is released. Without loss of generality A runs J_0 from time 0 until time 2^m . Stage 1 starts at time $b_1 = 2^m$. We maintain the invariant that at the start of a stage $i \geq 1$, A has not finished any jobs released in the previous stages, and A knows no lower bound on the remaining processing time of the jobs released in the previous stages.

We now describe stage $i \geq 1$. At the start of stage i a job J_i of class $c_i = c_{i-1} - \lg(4s(i+1))$ is released. Stage $i+1$ begins at the first time that A has run J_i for at least 2^{c_i} time units. We now argue $b_{i+1} \leq b_i + 2s(i+1)2^{c_i}$. Assume to reach a contradiction that this is not the case. We then set $p_i = 2^{c_i}$. The stretch of J_i is at least the length of stage i , which is at least $2s(i+1)2^{c_i}$, divided by p_i . That is, the stretch of J_i is at least $2s(i+1)$. Since $i+1$ jobs have been released by stage i , the average stretch is then at least $2s$. This contradicts the assumption that A is s -competitive with respect to average stretch.

In order to be able to continue this construction indefinitely we need to argue that the duration $b_{i+1} - b_i$ of stage i is at most $2^{c_{i-1}}/2$. By the previous paragraph we know that $b_{i+1} - b_i < 2s(i+1)2^{c_i}$. Using the definition of c_i we can conclude that $b_{i+1} - b_i < 2s(i+1)2^{c_{i-1} - \lg(4s(i+1))}$. By algebraic simplification we get the desired result that $b_{i+1} - b_i < 2^{c_{i-1}}/2$.

The adversary's schedule is exactly the same as A 's schedule up until the last stage k . During stage k , while A is running J_k , the adversary finishes all of the jobs released in previous stages. We set the processing times of the jobs so that at the end of stage k , A has k jobs unfinished, each with p_k/k remaining processing time. By releasing jobs of length p_k/k every p_k/k time units over a long time period T , A 's total flow time will approach kT , while the adversary's total flow time will approach T . This contradicts the assumption that A is $O(1)$ competitive with respect to average flow time.

Acknowledgments. We want to thank Michael Bender, S.Muthukrishnan and Rajmohan Rajaram for sharing their algorithm with us. We also thank Nikhil Bansal and an anonymous referee for helpful suggestions.

References

1. B. Awerbuch, Y. Azar, S. Leonardi and O. Regev. "Minimizing the flow time without migration", ACM Symposim on Theory of Computing, pp. 198–205, 1999.
2. M. Bender, S. Muthukrishnan, and R. Rajaraman, "Improved algorithms for stretch scheduling", ACM/SIAM Symposium on Discrete Algorithms, 762–771, 2002.
3. L. Becchetti, and S. Leonardi, "Non-Clairvoyant Scheduling to Minimize the Average Flow Time on Single and Parallel Machines" ACM Symposium on Theorey of Computing, 2001.
4. P. Berman and C. Coulston, "Speed is more powerful than clairvoyance", *Nordic Journal of Computing*, **6**(2), 181–193, 1999.
5. M. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers", in Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems (USITS '99), October 1999.
6. J. Edmonds, "Scheduling in the dark", *Theoretical Computer Science*, **235**(1), 109–141, 2000.
7. B. Kalyanasundaram, and K. Pruhs, "Speed is as powerful as clairvoyance", *Journal of the ACM*, **47**(4), 617–643, 2000.
8. B. Kalyanasundaram, and K. Pruhs, "Minimizing flow time nonclairvoyantly", IEEE Symposium on Foundations of Computer Science, 1997.
9. S. Leonardi, "A simpler proof of preemptive total flow time approximation on parallel machines", To appear in *Approximation and Online Algorithms*, E. Bampis eds, Lecture Notes in computer Science, Springer-Verlag, 2003. <http://www.dis.uniroma1.it/~leon/>
10. S. Leonardi and D. Raz. "Approximating total flow time on parallel machines", ACM Symposium on Theory of Computing, 110–119, El Paso, Texas, 1997.
11. R. Motwani, S. Phillips, and E. Torng, "Non-clairvoyant scheduling", *Theoretical Computer Science*, **130**, 17–47, 1994.
12. S. Muthukrishnan, R. Rajaraman, R. Shaheen, and J. Gehrke, "Online scheduling to minimize average stretch", IEEE Symposium on Foundations of Compter Science, 433–442, 1997.
13. C. Phillips, C. Stein, E. Torng, and J. Wein "Optimal time-critical scheduling via resource augmentation", *Algorithmica*, **32**, 163–200, 2002.

Algorithms for Graph Rigidity and Scene Analysis

Alex R. Berg¹ and Tibor Jordán^{2*}

¹ BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation), University of Aarhus, Ny Munkegade, building 540, Aarhus C, Denmark. aberg@brics.dk

² Department of Operations Research, Eötvös University, Pázmány sétány 1/C, 1117 Budapest, Hungary. jordan@cs.elte.hu

Abstract. We investigate algorithmic questions and structural problems concerning graph families defined by ‘edge-counts’. Motivated by recent developments in the unique realization problem of graphs, we give an efficient algorithm to compute the rigid, redundantly rigid, M -connected, and globally rigid components of a graph. Our algorithm is based on (and also extends and simplifies) the idea of Hendrickson and Jacobs, as it uses *orientations* as the main algorithmic tool.

We also consider families of bipartite graphs which occur in parallel drawings and scene analysis. We verify a conjecture of Whiteley by showing that $2d$ -connected bipartite graphs are d -tight. We give a new algorithm for finding a maximal d -sharp subgraph. We also answer a question of Imai and show that finding a maximum size d -sharp subgraph is NP-hard.

1 Introduction

A d -dimensional *framework* is a straight line embedding of a graph $G = (V, E)$ in the d -dimensional Euclidean space. We may think of the edges as rigid bars and vertices as rotatable joints, and say that the framework is ‘rigid’ if it has no non-trivial deformations (for precise definitions, in terms of the ‘rigidity matrix’ of the framework, see [18]). If the coordinates are ‘generic’ then rigidity depends only on the graph. Characterising rigidity of frameworks (and graphs) is an old problem in statics (and graph theory). In two dimensions there is a combinatorial characterisation, see Theorem 2.

Rigidity of graphs plays an interesting role in unique graph realizations. Two frameworks of the same graph G are *equivalent* if corresponding edges of the two frameworks have the same length, and they are *congruent* if the distance between all corresponding pairs of vertices is the same. We say that a framework is a *unique realization* of G in R^d if every equivalent framework of G is also congruent.

* Supported by the Hungarian Scientific Research Fund no. T037547, F034930 and FKFP grant no. 0143/2001. Part of this research was done while the second author visited BRICS, University of Aarhus, Denmark.

Hendrickson [6] proved that if G has a unique generic realization then G is $(d+1)$ -connected and redundantly rigid (that is, $G - e$ is rigid for all $e \in E(G)$). See also [2]. It was proved in [10] that for $d = 2$ these conditions are sufficient to guarantee that every generic framework of G is a unique realization, in which case the graph is called *globally rigid*. Thus unique realizability is a generic property in two dimensions. This recent result motivated us to give an algorithm for finding the finer structure of rigid, redundantly rigid, and M -connected components of a graph in two dimensions (see Section 3 for the definitions), and for identifying the maximal globally rigid subgraphs of G .

The basic algorithmic questions concerning rigidity were answered many years ago. There exist polynomial algorithms for testing rigidity as well as for computing the ‘degree of freedom’ in $O(n^2)$ time, where n is the number of vertices of G . The algorithms of Sugihara [14] and Hendrickson [6] used techniques from matching theory, while Imai [9] used network flows. Gabow and Westermann [4] used matroid sums and showed that the M -connected components can also be found in $O(n^2)$ time.

Our algorithm also runs in $O(n^2)$ time and in some sense it is similar to the previous algorithms. However, it requires no auxiliary graphs or digraphs, or matroids, to work on. Thus it is perhaps simpler and easier to visualize. It works with *orientations* of G and only performs reachability searches (and reorients directed paths). The idea of such an algorithm is due to Hendrickson and Jacobs [7], who gave a basic version of this algorithm as a so-called “pebble game” (see also [11]) for finding the rigid components in $O(n^2)$ time. We simplify the terminology of [7] and make it more suitable for finding other substructures as well as for extensions to other families of graphs defined by edge counts.

A different area of discrete applied geometry, where such families occur, is parallel drawings. A *polyhedral incidence structure* is a bipartite graph $S = (V, F; I)$ where the colour classes V and F represent the *vertices* and *faces*, respectively, and the edge set I represents the vertex-face incidences. A d -*scene* of S assigns points of R^d to the vertices and hyperplanes to the faces such that all the incidence constraints are satisfied. For a set of generic normals for the faces, one may ask whether there is a d -scene of S with the given normals. The existence of a such a (non-trivial) d -scene depends only on S . Whiteley [16,17] characterised the bipartite graphs whose edge set is a base in the corresponding matroid (called *minimally d -tight* graphs) as well as the graphs for which such a d -scene exists with distinct points assigned to distinct vertices (called *d -sharp*). A minimally d -tight graph satisfies the count $|I| = d|V| + |F| - d$ and $|I'| \leq d|V(I')| + |F(I')| - d$, for all $\emptyset \neq I' \subseteq I$, where $V(I')$ and $F(I')$ denotes the number of vertices induced by I' in V and F , respectively. A d -sharp graph has $|I'| \leq d|V(I')| + |F(I')| - d$, for all $I' \subseteq I$ with $|V(I')| \geq 2$. Whiteley conjectured that every $2d$ -connected incidence graph has a minimally d -tight spanning subgraph [18, p.211]. We prove this conjecture.

We also investigate d -sharp graphs, but in a different context. The basic problem of scene analysis of polyhedral pictures is how to reconstruct a 3-dimensional polyhedron from a given planar projection, a so-called line drawing, by setting

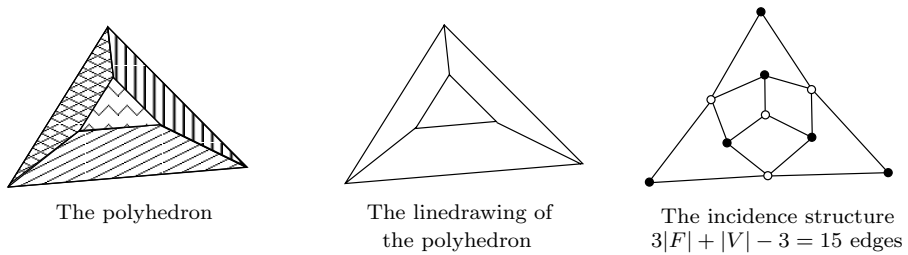


Fig. 1. A line drawing with a minimally 3-tight incidence graph.

the third coordinates of the vertices so that vertices that belong to the same face in the line drawing are coplanar (and such that each pair of faces sharing a vertex have distinct planes). Assuming generic points for the line drawing Sugihara [15] and Whiteley [17] characterised the reconstructible line drawings in terms of their bipartite incidence graph $P = (V, F; I)$, which encodes the vertex-face incidences. This characterisation turns out to be ‘polar’ to parallel drawings: there is a proper ‘lifting’ for P if and only if the bipartite graph obtained from P by interchanging the role of V and F is 3-sharp. Due to numerical errors, line drawings produced by computers are ‘generic’, and may produce non-reconstructible pictures. To cope with this problem Sugihara [15] developed an algorithm which attempts to reconstruct the polyhedron from a maximal 3-sharp subgraph of the incidence graph of the line drawing, see also Imai [9]. To illustrate the applicability of the orientation based approach to other edge counts, we give a different algorithm for finding a maximal d -sharp subgraph. Furthermore, answering an open question posed by Imai [9], we show that finding a maximum size d -sharp subgraph is NP-hard for $d \geq 2$.

2 Orientations with Upper Bounds on the In-Degrees

Let $G = (V, E)$ be a graph. An *orientation* $D = (V, A)$ of G is a directed graph obtained from G by replacing each edge uv by a directed edge (directed from u to v or from v to u). For a subset $X \subseteq V$ let $G[X]$ denote the subgraph of G induced by X , and let $i_G(X)$ (or simply $i(X)$) denote the number of edges in $G[X]$. If $D = (V, A)$ is a directed graph and $X \subseteq V$ then $\rho_D(X)$ denotes the number of directed edges entering X . This is the *in-degree* of X . The in-degree of a vertex v is denoted by $\rho_D(v)$. Let $g : V \rightarrow \mathbb{Z}_+$ assign non-negative integers to the vertices of G . For $X \subseteq V$ we use the notation $g(X) := \sum_{v \in X} g(v)$. We say that an orientation D of G is a *g -orientation* if $\rho_D(v) \leq g(v)$ holds for all $v \in V$.

The next result, due to Frank and Gyárfás, characterises when G has an orientation satisfying given upper bounds on the in-degrees. We present its simple proof, since it illustrates the basic steps of our algorithm: searching for reachable vertices and reorienting directed paths.

Theorem 1. [3] *Let $G = (V, E)$ be a graph and $g : V \rightarrow \mathbb{Z}_+$. Then G has a g -orientation if and only if*

$$i(X) \leq g(X) \text{ for all } X \subseteq V. \quad (1)$$

Proof. To see necessity suppose that D is a g -orientation of G and let $X \subseteq V$. Then $i(X) = \sum_{v \in X} \rho_D(v) - \rho_D(X) \leq g(X)$.

To prove sufficiency suppose that (1) holds and choose an orientation D' of G for which $h(D') := \sum_{v \in V} (\rho(v) - g(v))^+$ is as small as possible, where $x^+ := \max\{x, 0\}$ for some integer x . If $h(D') = 0$ then D' is a g -orientation. Otherwise there is a vertex s with $\rho_{D'}(s) > g(s)$. Let S denote the set of vertices from which there is a directed path to s in D' . Clearly, $\rho_{D'}(S) = 0$. If there is a vertex $t \in S$ with $\rho_{D'}(t) < g(t)$ then by reorienting the edges of a directed path from t to s we obtain an orientation D'' with $h(D'') = h(D') - 1$, contradicting the choice of D' . Thus we have $\rho_{D'}(v) \geq g(v)$ for each vertex $v \in S$, and hence, since $\rho_{D'}(s) > g(s)$, we obtain $i(S) = \sum_{v \in S} \rho_{D'}(v) - \rho_{D'}(S) > \sum_{v \in S} g(v) = g(S)$, contradicting (1). This proves the theorem. $\square$

This proof leads to an algorithm for finding a g -orientation, if exists. It shows that if (1) holds then any orientation D' of G can be turned into a g -orientation by finding and reorienting directed paths $h(D')$ times. Such an elementary step (which decreases h by one) can be done in linear time.

3 Rigid Graphs and the Rigidity Matroid

The following combinatorial characterization of two-dimensional rigidity is due to Laman. A graph G is said to be *minimally rigid* if G is rigid, and $G - e$ is not rigid for all $e \in E$. A graph is *rigid* if it has a minimally rigid spanning subgraph.

Theorem 2. [12] *$G = (V, E)$ is minimally rigid if and only if $|E| = 2|V| - 3$ and*

$$i(X) \leq 2|X| - 3 \text{ for all } X \subseteq V \text{ with } |X| \geq 2. \quad (2)$$

In fact, Theorem 2 characterises the bases of the *rigidity matroid* of the complete graph on vertex set V . In this matroid a set of edges S is *independent* if the subgraph induced by S satisfies (2). The *rigidity matroid* of G , denoted by $\mathcal{M}(G) = (E, \mathcal{I})$, is the restriction of the rigidity matroid of the complete graph to E . Thus G is rigid if and only if E has rank $2|V| - 3$ in $\mathcal{M}(G)$. If G is rigid and $H = (V, E')$ is a spanning subgraph of G , then H is minimally rigid if and only if E' is a base in $\mathcal{M}(G)$.

3.1 A Base, the Rigid Components, and the Rank

To test whether G is rigid (or more generally, to compute the rank of $\mathcal{M}(G)$) we need to find a base of $\mathcal{M}(G)$. This can be done greedily, by building up a maximal independent set by adding (or rejecting) edges one by one. The key of

this procedure is the independence test: given an independent set I and an edge e , check whether $I + e$ is independent. With Theorem 1 we can do this in linear time as follows (see also [7]).

Let $g_2 : V \rightarrow Z_+$ be defined by $g_2(v) = 2$ for all $v \in V$. For two vertices $u, v \in V$ let $g_2^{uv} : V \rightarrow Z_+$ be defined by $g_2^{uv}(u) = g_2^{uv}(v) = 0$, and $g_2^{uv}(w) = 2$ for all $w \in V - \{u, v\}$.

Lemma 1. *Let $I \subset E$ be independent and let $e = uv$ be an edge, $e \in E - I$. Then $I + e$ is independent if and only if (V, I) has a g_2^{uv} -orientation.*

Proof. Let $H = (V, I)$ and $H' = (V, I + e)$. First suppose that $I + e$ is not independent. Then there is a set $X \subseteq V$ with $i_{H'}(X) \geq 2|X| - 2$. Since I is independent, we must have $u, v \in X$ and $i_H(X) = 2|X| - 3$. Hence $i_H(X) = 2|X| - 3 > g_2^{uv}(X) = 2|X| - 4$, showing that H has no g_2^{uv} -orientation.

Conversely, suppose that $I + e$ is independent, but H has no g_2^{uv} -orientation. By Theorem 1 this implies that there is a set $X \subseteq V$ with $i_H(X) > g_2^{uv}(X)$. Since $i_H(X) \leq 2|X| - 3$ and $g_2^{uv}(X) = 2|X| - 2|X \cap \{u, v\}|$, this implies $u, v \in X$ and $i_H(X) = 2|X| - 3$. Then $i_{H'}(X) = 2|X| - 2$, contradicting the fact that $I + e$ is independent. $\square$

A weak g_2^{uv} -orientation D of G satisfies $\rho_D(w) \leq 2$ for all $w \in V - \{u, v\}$ and has $\rho_D(u) + \rho_D(v) \leq 1$. It follows from the proof that a weak g_2^{uv} -orientation of (V, I) always exists.

If we start with a g_2 -orientation of $H = (V, I)$ then the existence of a g_2^{uv} -orientation of H can be checked by at most four elementary steps (reachability search and reorientation) in linear time. Note also that H has $O(n)$ edges, since I is independent.

This gives rise to a simple algorithm for computing the rank of E in $\mathcal{M}(G)$. By maintaining a g_2 -orientation of the subgraph of the current independent set I , testing an edge needs only $O(n)$ time, and hence the total running time is $O(nm)$, where $m = |E|$. We shall improve this to $O(n^2)$ by identifying large rigid subgraphs.

We say that a maximal rigid subgraph of G is a *rigid component* of G . Clearly, every edge belongs to some rigid component, and rigid components are induced subgraphs. Since the union of two rigid subgraphs sharing an edge is also rigid, the edge sets of the rigid components partition E .

We can maintain the rigid components of the set of edges considered so far as follows. Let I be an independent set, let $e = uv$ be an edge with $e \in E - I$, and suppose that $I + e$ is independent. Let D be a g_2^{uv} -orientation of (V, I) . Let $X \subseteq V$ be the maximal set with $u, v \in X$, $\rho_D(X) = 0$, and such that $\rho_D(x) = 2$ for all $x \in X - \{u, v\}$. Clearly, such a set exists, and it is unique. It can be found by identifying the set $V_1 = \{x \in V - \{u, v\} : \rho_D(x) \leq 1\}$, finding the set $\hat{V}_1$ of vertices reachable from V_1 in D , and then taking $X = V - \hat{V}_1$. The next lemma shows how to update the set of rigid components when a new edge e is added to I .

Lemma 2. *Let $H' = (V, I + e)$. Then $H'[X]$ is a rigid component of H' .*

Thus, when we add e to I , the set of rigid components is updated by adding $H'[X]$ and deleting each component whose edge set is contained by the edge set of $H'[X]$. Maintaining this list can be done in linear time. Furthermore, we can reduce the total running time to $O(n^2)$ by performing the independence test for $I + e$ only if e is not spanned by any of the rigid components on the current list (and otherwise rejecting e , since $I + e$ is clearly dependent).

3.2 The M -Circuits and the Redundantly Rigid Components

Given a graph $G = (V, E)$, a subgraph $H = (W, C)$ is said to be an M -circuit in G if C is a circuit (i.e. a minimal dependent set) in $\mathcal{M}(G)$. G is an M -circuit if E is a circuit in $\mathcal{M}(G)$. By using (2) one can deduce the following properties.

Lemma 3. *Let $G = (V, E)$ be a graph without isolated vertices. Then G is an M -circuit if and only if $|E| = 2|V| - 2$ and $G - e$ is minimally rigid for all $e \in E$.*

A subgraph $H = (W, F)$ is *redundantly rigid* if H is rigid and $H - e$ is rigid for all $e \in F$. M -circuits are redundantly rigid by Lemma 3(b). A *redundantly rigid component* is either a maximal redundantly rigid subgraph of G (in which case the component is *non-trivial*) or a subgraph consisting of a single edge e , when e is contained in no redundantly rigid subgraph of G (in which case it is *trivial*). The redundantly rigid components are induced subgraphs and their edge sets partition the edge set of G . See Figure 2 for an example. An edge $e \in E$ is a *bridge* if e belongs to all bases of $\mathcal{M}(G)$. It is easy to see that each bridge e is a trivial redundantly rigid component. Let $B \subseteq E$ denote the set of bridges in G . The key to finding the redundantly rigid components efficiently is the following lemma.

Lemma 4. *The set of non-trivial redundantly rigid components of G is equal to the set of rigid components of $G' = (V, E - B)$.*

Thus we can identify the redundantly rigid components of G by finding the bridges of G and then finding the rigid components of the graph $G - B$.

3.3 The M -Connected Components and Maximal Globally Rigid Subgraphs

Given a matroid $\mathcal{M} = (E, \mathcal{I})$, one can define a relation on E by saying that $e, f \in E$ are related if $e = f$ or there is a circuit C in $\mathcal{M}$ with $e, f \in C$. It is well-known that this is an equivalence relation. The equivalence classes are called the *components* of $\mathcal{M}$. If $\mathcal{M}$ has at least two elements and only one component then $\mathcal{M}$ is said to be *connected*. Note that the trivial components (containing only one element) of $\mathcal{M}$ are exactly the bridges of G .

We say that a graph $G = (V, E)$ is M -connected if $\mathcal{M}(G)$ is connected. The M -connected components of G are the subgraphs of G induced by the components of $\mathcal{M}(G)$. The M -connected components are also edge-disjoint induced subgraphs. They are redundantly rigid.

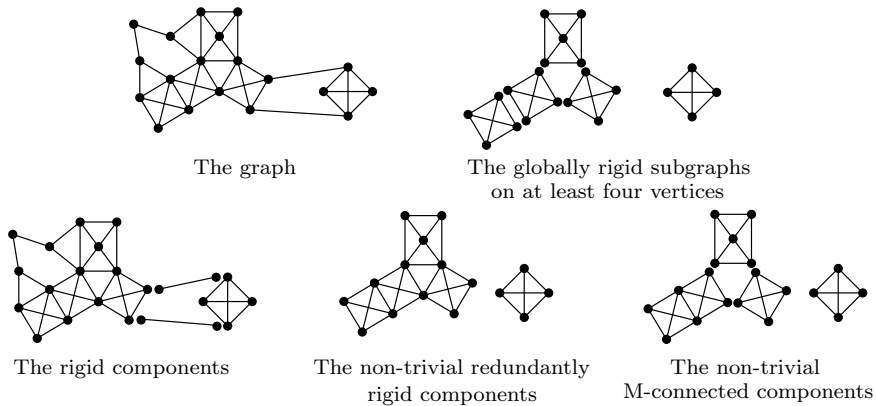


Fig. 2. Decompositions of a graph.

To find the bridges and M -connected components we need the following observations. Suppose that I is independent but $I + e$ is dependent. The *fundamental circuit* of e with respect to I is the (unique) circuit contained in $I + e$. Our algorithm will also identify a set of fundamental circuits with respect to the base I that it outputs. To find the fundamental circuit of $e = uv$ with respect to I we proceed as follows. Let D be a weak g_2^{uv} -orientation of (V, I) (with $\rho_D(v) = 1$, say). As we noted earlier, such an orientation exists. Let $Y \subseteq V$ be the (unique) minimal set with $u, v \in Y$, $\rho_D(Y) = 0$, and such that $\rho_D(x) = 2$ for all $x \in Y - \{u, v\}$. This set exists, since $I + e$ is dependent. Y is easy to find: it is the set of vertices that can reach v in D .

Lemma 5. *The edge set induced by Y in $(V, I + e)$ is the fundamental circuit of e with respect to I .*

Thus if $I + e$ is dependent, we can find the fundamental circuit of e in linear time. Our algorithm will maintain a list of M -connected components and compute the fundamental circuit of $e = uv$ only if u and v are not in the same M -connected component. Otherwise e is classified as a non-bridge edge. When a new fundamental circuit is found, its subgraph will be merged into one new M -connected component with all the current M -connected components whose edge set intersects it. It can be seen that the final list of M -connected components will be equal to the set of M -connected components of G , and the edges not induced by any of these components will form the set of bridges of G . It can also be shown that the algorithm computes $O(n)$ fundamental circuits, so the total running time is still $O(n^2)$. The algorithm can also determine an *ear-decomposition* of $\mathcal{M}(G)$ (see [10]), for an M -connected graph G , within the same time bound.

Thus to identify the maximal globally rigid subgraphs on at least four vertices we need to search for the maximal 3-connected subgraphs of the M -connected

components of G . This can be done in linear time by using the algorithm of Hopcroft and Tarjan [8] which decomposes the graph into its 3-connected blocks.

4 Tight and Sharp Bipartite Graphs

Let $G = (A, B; E)$ be a bipartite graph. For subsets $W \subseteq A \cup B$ and $F \subseteq E$ let $W(F)$ denote the set of those vertices of W which are incident to edges of F . We say that G is *minimally d -tight* if $|E| = d|A| + |B| - d$ and for all $\emptyset \neq E' \subseteq E$ we have

$$|E'| \leq d|A(E')| + |B(E')| - d. \quad (3)$$

G is called *d -tight* if it has a minimally d -tight spanning subgraph. It is not difficult to show that the subsets $F \subseteq E$ for which every $\emptyset \neq E' \subseteq F$ satisfies (3) form the independent sets of a matroid on groundset E . By calculating the rank function of this matroid we obtain the following characterization.

Theorem 3. [17] $G = (A, B; E)$ is d -tight if and only if

$$\sum_{i=1}^t (d \cdot |A(E_i)| + |B(E_i)| - d) \geq d|A| + |B| - d \quad (4)$$

for all partitions $\mathcal{E} = \{E_1, E_2, \dots, E_t\}$ of E .

4.1 Highly Connected Graphs Are d -Tight

Lovász and Yemini [13] proved that 6-connected graphs are rigid. A similar result, stating that $2d$ -connected bipartite graphs are d -tight, was conjectured by Whiteley [16,18]. We prove this conjecture by using an approach similar to that of [13]. We say that a graph $G = (V, E)$ is *k -connected in W* , where $W \subseteq V$, if there exist k openly disjoint paths in G between each pair of vertices of W .

Theorem 4. Let $G = (A, B; E)$ be $2d$ -connected in A , for some $d \geq 2$, and suppose that there is no isolated vertex in B . Then G is d -tight.

Proof. For a contradiction suppose that G is not d -tight. By Theorem 3 this implies that there is a partition $\mathcal{E} = \{E_1, E_2, \dots, E_t\}$ of E with $\sum_{i=1}^t (d \cdot |A(E_i)| + |B(E_i)| - d) < d|A| + |B| - d$. Since G is $2d$ -connected in A and there is no isolated vertex in B , we have $d|A(E)| + |B(E)| - d = d|A| + |B| - d$. Thus $t \geq 2$ must hold.

Claim. Suppose that $A(E_i) \cap A(E_j) \neq \emptyset$ for some $1 \leq i < j \leq t$. Then $d|A(E_i)| + |B(E_i)| - d + d|A(E_j)| + |B(E_j)| - d \geq d|A(E_i \cup E_j)| + |B(E_i \cup E_j)| - d$.

The claim follows from the inequality: $d|A(E_i)| + |B(E_i)| - d + d|A(E_j)| + |B(E_j)| - d = d|A(E_i) \cup A(E_j)| + d|A(E_i) \cap A(E_j)| + |B(E_i) \cup B(E_j)| + |B(E_i) \cap B(E_j)| - 2d \geq d|A(E_i \cup E_j)| + |B(E_i \cup E_j)| - d$, where we used $d|A(E_i) \cap A(E_j)| \geq d$ (since $A(E_i) \cap A(E_j) \neq \emptyset$), and $|B(E_i) \cap B(E_j)| \geq 0$.

By the Claim we can assume that $A(E_i) \cap A(E_j) = \emptyset$ for all $1 \leq i < j \leq t$. Let $B' \subseteq B$ be the set of those vertices of B which are incident to edges from at least two classes of partition $\mathcal{E}$. Since $A(E_i) \cap A(E_j) = \emptyset$ for all $1 \leq i < j \leq t$, and $t \geq 2$, the vertex set $B'(E_i)$ separates $A(E_i)$ from $\cup_{j \neq i} A(E_j)$ for all $E_i \in \mathcal{E}$. Hence, since G is $2d$ -connected in A , we must have

$$|B'(E_i)| \geq 2d \text{ for all } 1 \leq i \leq t. \quad (5)$$

To finish the proof we count as follows. Since $A(E_i) \cap A(E_j) = \emptyset$ for all $1 \leq i < j \leq t$, we have $\sum_1^t |A(E_i)| = |A|$. Hence $\sum_1^t (|B(E_i)| - d) < |B| - d$ follows, which gives

$$\sum_1^t (|B'(E_i)| - d) < |B'| - d. \quad (6)$$

Furthermore, it follows from (5) and the definition of B' that for every vertex $b \in B'$ we have

$$\sum_{E_i: b \in B(E_i)} \left(1 - \frac{d}{|B'(E_i)|}\right) \geq 2\left(1 - \frac{d}{2d}\right) = 1.$$

Thus

$$|B'| \leq \sum_{b \in B'} \sum_{E_i: b \in B(E_i)} \left(1 - \frac{d}{|B'(E_i)|}\right) = \sum_1^t |B'(E_i)| \left(1 - \frac{d}{|B'(E_i)|}\right) = \sum_1^t (|B'(E_i)| - d),$$

which contradicts (6). This proves the theorem. $\square$

4.2 Testing Sharpness and Finding Large Sharp Subgraphs

By modifying the count in (3) slightly, we obtain a family of bipartite graphs which plays a central role in scene analysis (for parameter $d = 3$). We say that a bipartite graph $G = (A, B; E)$ is d -sharp, for some integer $d \geq 1$, if

$$|E'| \leq d|A(E')| + |B(E')| - (d + 1) \quad (7)$$

holds for all $E' \subseteq E$ with $|A(E')| \geq 2$. A set $F \subseteq E$ is d -sharp if it induces a d -sharp subgraph.

As it was pointed out by Imai [9], the count in (7) does not always define a matroid on the edge set of G . Hence to test d -sharpness one cannot directly apply the general framework which works well for rigidity and d -tightness. Sugihara [15] developed an algorithm for testing 3-sharpness and, more generally, for finding a maximal 3-sharp subset of E . Imai [9] improved the running time to $O(n^2)$. Their algorithms are based on network flow methods.

An alternative approach is as follows. Let us call a maximal d -tight subgraph of G a d -tight component. As in the case of rigid components, one can show that the d -tight components are pairwise edge-disjoint and their edge sets partition E . Moreover, by using the appropriate version of our orientation based algorithm,

they can be identified in $O(n^2)$ time. The following lemma shows how to use these components to test d -sharpness (and to find a maximal d -sharp edge set) in $O(n^2)$ time.

Lemma 6. *Let $G = (A, B; E)$ be a bipartite graph and $d \geq 1$ be an integer. Then G is d -sharp if and only if each d -tight component H satisfies $|V(H) \cap A| = 1$.*

Proof. Necessity is clear from the definition of d -tight and d -sharp graphs. To see sufficiency suppose that each d -tight component H satisfies $|V(H) \cap A| = 1$, but G is not d -sharp. Then there exists a set $I \subseteq E$ with (i) $|A(I)| \geq 2$ and (ii) $|I| \geq d|A(I)| + |B(I)| - d$. Let I be a minimal set satisfying (i) and (ii). Suppose that I satisfies (ii) with strict inequality and let $e \in I$ be an edge. By the minimality of I the set $I - e$ must violate (i). Thus $|A(I - e)| = 1$. By (i), and since $d \geq 2$, this implies $|I| = |I - e| + 1 = |B(I - e)| + 1 \leq |B(I)| + 1 \leq d|A(I)| + |B(I)| - d$, a contradiction. Thus $|I| = d|A(I)| + |B(I)| - d$. The minimality of I (and the fact that each set I' with $|A(I')| = 1$ trivially satisfies $|I'| = d|A(I')| + |B(I')| - d$) implies that (3) holds for each non-empty subset of I . Thus I induces a d -tight subgraph H' , which is included by a d -tight component H with $|V(H) \cap A| \geq |A(I)| \geq 2$, contradicting our assumption. $\square$

Imai [9] asked whether a maximum size 3-sharp edge set of G can be found in polynomial time. We answer this question by showing that the problem is NP-hard.

Theorem 5. *Let $G = (A, B; E)$ be a bipartite graph, and let $d \geq 2$, $N \geq 1$ be integers. Deciding whether G has a d -sharp edge set $F \subseteq E$ with $|F| \geq N$ is NP-complete.*

Proof. We shall prove that the NP-complete VERTEX COVER problem can be reduced to our decision problem. Consider an instance of VERTEX COVER, which consists of a graph $D = (V, J)$ and an integer M (and the question is whether D has a vertex cover of size at most M). Our reduction is as follows. First we construct a bipartite graph $H_0 = (A_0, B_0; E_0)$, where $A_0 = \{c\}$, $B_0 = V$ corresponds to the vertex set of D , and $E_0 = \{cu : u \in B_0\}$. We call c the center of H_0 . Thus H_0 is a star which spans the vertices of D from a new vertex c at the center. The bipartite graph $H = (A, B; E)$ that we construct next is obtained by adding a clause to H_0 for each edge of D . The clause for an edge $e = uv \in J$ is the following (see Figure 3). $f_1, f_2, \dots, f_{d-2}$ are the new vertices of B and x is the new vertex of A in the clause (these vertices do not belong to other clauses). The edges of the clause are ux, vx and $f_i x, f_i c$ for $i \in \{1, 2, \dots, d-2\}$ (so if $d = 2$ the only new edges are ux and vx). Let C'_e denote these edges of the clause. So $C'_e \cap C'_f = \emptyset$ for each pair of distinct edges $e, f \in J$. Let $C_{uv} := C'_{uv} + cu + cv$. Note that C_e is not d -sharp, since $|A(C_e)| = 2$ and $2d = |C_e| > d|A(C_e)| + |B(C_e)| - (d+1) = 2d - 1$. However, it is easy to check that removing any edge makes C_e d -sharp. We set $N = |E| - M$.

Lemma 7. *Let $Y \subseteq E$. Then $E - Y$ is d -sharp if and only if $C_e \cap Y \neq \emptyset$ for every $e \in J$.*

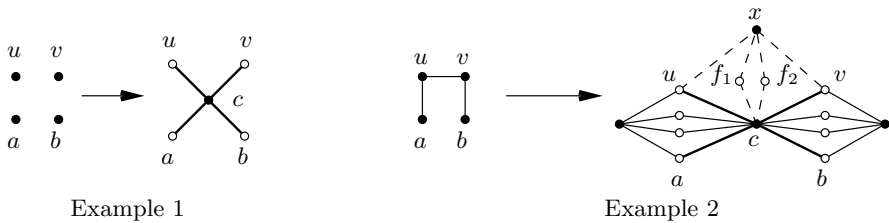


Fig. 3. Two examples of the reduction for $d = 4$. Empty circles are the vertices of B , filled circles are the vertices of A of H . The dotted lines are the edges of C'_{uv} . The thick lines are the edges of H_0 .

Proof. The only if direction follows from the fact that C_e is not d -sharp for $e \in J$. To see the other direction suppose, for a contradiction, that $Z \subseteq E - Y$ is a set with $|A(Z)| \geq 2$ and $|Z| > d|A(Z)| + |B(Z)| - (d + 1)$. We may assume that, subject to these properties, $A(Z) \cup B(Z)$ is minimal.

Minimality implies that if $|A(Z)| \geq 3$ then each vertex $w \in A(Z)$ is incident to at least $d + 1$ edges of Z . Thus, since every vertex of $A - c$ has degree d in H , we must have $|A(Z)| = 2$. Minimality also implies that each vertex $f \in B(Z)$ is incident to at least two edges of Z . If $c \in A(Z)$ then $Z \subseteq C_e$, for some $e \in J$, since each vertex $f \in B(Z)$ has at least two edges from Z . But then $Y \cap C_e \neq \emptyset$ implies that Z is d -sharp. On the other hand if $c \notin A(Z)$ and $|A(Z)| = 2$ then $|Z| \leq 2$, and hence Z is d -sharp. This contradicts the choice of Z . Thus $E - Y$ is d -sharp. $\square$

Lemma 8. *Let $Y \subseteq E$ and suppose that $E - Y$ is d -sharp. Then there is a set $Y' \subseteq E$ with $|Y'| \leq |Y|$ for which $E - Y'$ is d -sharp and $Y' \subseteq \{cu : u \in B\}$.*

Proof. Since $E - Y$ is d -sharp and C_e is not d -sharp we must have $C_e \cap Y \neq \emptyset$ for each $e \in J$. We obtain Y' by modifying Y with the following operations. If $|C_{uv} \cap Y| \geq 2$ for some $uv \in J$ then we replace $C_{uv} \cap Y$ by $\{cu, cv\}$. If $C_{uv} \cap Y = \{f\}$ and $f \notin \{cu, cv\}$ for some $uv \in J$ then we replace f by cu in Y . The new set Y' satisfies $|Y'| \leq |Y|$, and, by Lemma 7, $E - Y'$ is also d -sharp. $\square$

We claim that H has a d -sharp edge set F with $|F| \geq N$ if and only if D has a vertex cover of size at most M . First suppose $F \subseteq E$ is d -sharp with $|F| \geq N$. Now $E - Y$ is d -sharp for $Y := E - F$, and hence, by Lemma 8, there is a set $Y' \subseteq E$ with $|Y'| \leq |Y| \leq M$ for which $E - Y'$ is d -sharp and $Y' \subseteq \{cu : u \in B\}$. Since $E - Y'$ is d -sharp, Lemma 7 implies that $X = \{u \in V : cu \in Y'\}$ is a vertex cover of D of size at most M .

Conversely, suppose that X is a vertex cover of D of size at most M . Let $Y = \{cu : u \in X\}$. Since X intersects every edge of D , we have $Y \cap C_{uv} \neq \emptyset$ for every $e \in J$. Thus, by Lemma 7, $F := E - Y$ is d -sharp, and $|F| \geq |E| - M = N$. Since our reduction is polynomial, this equivalence completes the proof of the theorem. $\square$

Note that finding a maximum size d -sharp edge set is easy for $d = 1$, since an edge set F is 1-sharp if and only if each vertex of B is incident to at most one edge of F .

References

1. A. BERG AND T. JORDÁN, A proof of Connelly's conjecture on 3-connected circuits of the rigidity matroid, *J. Combinatorial Theory, Ser. B.* Vol. 88, 77–97, 2003..
2. R. CONNELLY, On generic global rigidity, Applied geometry and discrete mathematics, 147–155, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 4, Amer. Math. Soc., Providence, RI, 1991.
3. A. FRANK AND A. GYÁRFÁS, How to orient the edges of a graph, Combinatorics, (Keszthely), Coll. Math. Soc. J. Bolyai 18, 353–364, North-Holland, 1976.
4. H.N. GABOW AND H.H. WESTERMANN, Forests, frames and games: Algorithms for matroid sums and applications, *Algorithmica* 7, 465–497 (1992).
5. J. GRAVER, B. SERVATIUS, AND H. SERVATIUS, Combinatorial Rigidity, AMS Graduate Studies in Mathematics Vol. 2, 1993.
6. B. HENDRICKSON, Conditions for unique graph realizations, *SIAM J. Comput.* **21** (1992), no. 1, 65–84.
7. B. HENDRICKSON AND D. JACOBS, An algorithm for two-dimensional rigidity percolation: the pebble game, *J. Computational Physics* 137, 346–365 (1997).
8. J.E. HOPCROFT AND R.E. TARJAN, Dividing a graph into triconnected components, *SIAM J. Comput.* 2 (1973), 135–158.
9. H. IMAI, On combinatorial structures of line drawings of polyhedra, *Discrete Appl. Math.* 10, 79 (1985).
10. B. JACKSON AND T. JORDÁN, Connected rigidity matroids and unique realizations of graphs, EGRES Technical Report 2002-12 (www.cs.elte.hu/egres/), submitted to J. Combin. Theory Ser. B.
11. D.J. JACOBS AND M.F. THORPE, Generic rigidity percolation: the pebble game, *Phys. Rev. Lett.* 75, 4051 (1995).
12. G. LAMAN, On graphs and rigidity of plane skeletal structures, *J. Engineering Math.* 4 (1970), 331–340.
13. L. LOVÁSZ AND Y. YEMINI, On generic rigidity in the plane, *SIAM J. Algebraic Discrete Methods* 3 (1982), no. 1, 91–98.
14. K. SUGIHARA, On some problems in the design of plane skeletal structures, *SIAM J. Algebraic Discrete Methods* 4 (1983), no. 3, 355–362.
15. K. SUGIHARA, Machine interpretation of line drawings, MIT Press, 1986.
16. W. WHITELEY, Parallel redrawing of configurations in 3-space, preprint, Department of Mathematics and Statistics, York University, North York, Ontario, 1987.
17. W. WHITELEY, A matroid on hypergraphs with applications in scene analysis and geometry, *Discrete Comput. Geometry* 4 (1989) 75–95.
18. W. WHITELEY, Some matroids from discrete applied geometry. Matroid theory (Seattle, WA, 1995), 171–311, Contemp. Math., 197, Amer. Math. Soc., Providence, RI, 1996.

Optimal Dynamic Video-on-Demand Using Adaptive Broadcasting

Therese Biedl¹, Erik D. Demaine², Alexander Golynski¹, Joseph D. Horton³,
Alejandro López-Ortiz¹, Guillaume Poirier¹, and Claude-Guy Quimper¹

¹ School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1,
Canada, {agolynski, alopez-o, cquimper, gpoirier, biedl}@uwaterloo.ca

² MIT Laboratory for Computer Science, 200 Technology Square, Cambridge, MA
02139, USA, edemaine@mit.edu

³ Faculty of Computer Science, University of New Brunswick, P.O. Box 4400,
Fredericton, N. B. E3B 5A3, Canada, jdh@unb.ca

Abstract. We consider the transmission of a movie over a broadcast network to support several viewers who start watching at arbitrary times, after a wait of at most t_{wait} minutes. A recent approach called harmonic broadcasting optimally solves the case of many viewers watching a movie using a constant amount of bandwidth. We consider the more general setting in which a movie is watched by an arbitrary number v of viewers, and v changes dynamically. A natural objective is to minimize the amount of resources required to achieve this task. We introduce two natural measures of resource consumption and performance—total bandwidth usage and maximum momentary bandwidth usage—and propose strategies which are optimal for each of them. In particular, we show that an adaptive form of pyramid broadcasting is optimal for both measures simultaneously, up to constant factors. We also show that the maximum throughput for a fixed network bandwidth cannot be obtained by any online strategy.

1 Introduction

Video-on-demand. A drawback of traditional TV broadcasting schemes is that the signal is sent only once and all viewers wishing to receive it must be listening at time of broadcast. To address this problem, viewers rely on recording devices (VCR, TiVo) that allow them to postpone viewing time by recording the program at time of broadcast for later use. A drawback of this solution is that the viewer must predict her viewing preferences in advance or else record every single program being broadcast, neither of which is practical. One proposed solution is to implement a video-on-demand (VoD) distribution service in which movies or TV programs are sent at the viewer's request. Considerable commercial interest has inspired extensive study in the networking literature [CP01,DSS94,EVZ00,EVZ01,JT97,JT98,ME+01,PCL98a,PCL98b,VI96,Won88] and most recently in SODA 2002 [ES02,B-NL02].

Previous approaches. The obvious approach—to establish a point-to-point connection between the provider and each viewer to send each desired show at

each desired time—is incredibly costly. *Pay-per-view* is a system in which the system broadcasts a selected set of titles and viewers make selections among the titles offered. Each movie that is, say, n minutes long gets broadcast over k channels at equally spaced intervals. The viewer then waits at most n/k minutes before she can start watching the movie. If k were sufficiently large, pay-per-view would become indistinguishable from video-on-demand from the viewer’s perspective. This property is known as *near video-on-demand* (nVoD) or simply VoD. In practice, movies are roughly 120 minutes long, which would require an impractical number of channels per movie for even a 5-minute wait time.

Viswanathan and Imielinski [VI96] observed that if viewers have specialized hardware available (such as a TiVo, DVD-R, or digital decoder standard with current cable setups), then it is possible to achieve video-on-demand with substantially lower bandwidth requirements. In practice, the physical broadcast medium is typically divided into *physical channels*, each of which has precisely the required bandwidth for broadcasting a single movie at normal play speed (real time). The idea is to simultaneously transmit different segments of a movie across several channels. The set-top device records the signals and collates the segments into viewing order.

Harmonic broadcasting. Juhn and Tseng [JT97] introduced the beautiful concept of *harmonic broadcasting* which involves dividing a movie into n equal sized *segments* of length t_{wait} . Throughout the paper, it is assumed that movies are encoded at a constant bitrate. The n segments are broadcast simultaneously and repeatedly, but at different rates; refer to Figure 1. Specifically, if we label the segments $S_1, \dots, S_n$ in order, then segment S_i is sent at a rate of $1/i$. In other words, we set up n virtual channels, where virtual channel C_i has the capacity of $1/i$ of a physical channel, and channel C_i simply repeats segment S_i over and over. Whenever a viewer arrives, the first i segments will have been broadcast after monitoring for $i + 1$ time *units* (one time unit is t_{wait} minutes), so the viewer can start playing as soon as the first segment has arrived. The maximum waiting time for a viewer in this scheme is t_{wait} minutes. The number of physical channels required by this scheme is the sum of the virtual channel capacity, $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$, which is the n th Harmonic number H_n . Asymptotically, $H_n \approx \ln n + \gamma + O(1/n)$ where $\gamma \approx 0.5572$ is Euler’s constant.

Pâris et al. [PCL98b] improved this scheme and gave more precise bounds on the required bandwidth and needed waiting time. Engebretsen and Sudan [ES02] proved that harmonic broadcasting is an optimal broadcasting scheme for one movie with a specified maximum waiting time. This analysis assumes that the movie is encoded at a constant bitrate, and that at every time interval $[it_{\text{wait}}, (i + 1)t_{\text{wait}}]$ $i \in \{0, 1, \dots, n - 1\}$, at least one viewer starts watching the movie. Hence, harmonic broadcasting is effective provided there are at least as many viewers as segments in the movie (e.g., 24 for a 120-minute movie and a 5-minute wait), but overkill otherwise.

Adaptive broadcasting. We introduce a family of *adaptive* broadcasting schemes which adapt to a dynamic number v of viewers, and use considerably less bandwidth for lower values of v .

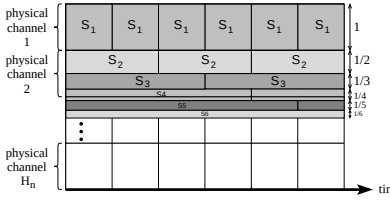


Fig. 1. Harmonic broadcasting.

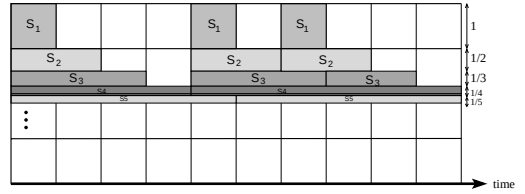


Fig. 2. Adaptive harmonic broadcasting.

To simplify the analysis, we decompose the entire broadcasting duration (which might be infinite) into *timespans* of $T = mt_{\text{wait}}$ minutes in length, (i.e., m segments long), and consider requests from some number v of viewers arriving within each such timespan. Notice that some segments sent to a viewer who started watching in timespan $[0, T]$ are actually broadcast in the next timespan $[T, 2T]$. We ignore the cost of such segments counting only those segments *received* during the current timespan. Provided that T is big enough, the ignored cost is negligible compared to the cost induced by non-overlapping viewers. Let v denote the number of viewers in a timespan, ignoring any leftover viewers who started watching in the previous timespan. The number v can change from timespan to timespan, and thus bounds stated in terms of v for a single timespan adapt to changes in v . The bounds we obtain can also be combined to apply to longer time intervals: in a bound on total bandwidth, the v term becomes the average value of v , and in a bound on maximum bandwidth used at any time, the v term becomes the maximum value of v over the time interval.

The viewer arrival times are unknown beforehand, and hence the algorithm must react in an online fashion. In this scenario, our goal is to minimize bandwidth required to support VoD for v viewers, where the maximum waiting time is a fixed parameter t_{wait} . Such an algorithm must adapt to changing (and unknown) values of v , and adjust its bandwidth usage according. In particular, it is clear that in general harmonic broadcasting is suboptimal, particularly for small values of v . Carter et al. [CP01] introduced this setting of a variable number of viewers, and proposed a heuristic for minimizing bandwidth consumption; here we develop algorithms that are guaranteed to be optimal.

Objectives. For a given sequence of viewer arrival times, we propose three measures of the efficiency of an adaptive broadcasting strategy:

1. Minimizing the total amount of data transmitted, which models the requirement of a content provider who purchases capacity by the bit from a data carrier. The total capacity required on the average is also a relevant metric if we assume a large number of movies being watched with requests arriving randomly and independently.
2. Minimizing the maximum number of channels in use at any time, which models the realistic constraint that the available bandwidth has a hard upper limit imposed by hardware, and that bandwidth should be relatively balanced throughout the transmission.

3. Obtaining a feasible schedule subject to a fixed bandwidth bound, which models the case where the content provider pays for a fixed amount of bandwidth, whether used or not, and wishes to maximize the benefit it derives from it. (In contrast to the previous constraints, this measure favors early broadcasting so that the bandwidth is always fully used.)

Broadcasting schemes can be distinguished according to two main categories: *integral* which distribute entire segments of movies from beginning to end in physical channels, in real time, and *nonintegral* which distribute segments at various nonintegral rates and allow viewers to receive segments starting in the middle. For example, pay-per-view is a simple integral scheme, whereas harmonic broadcasting is nonintegral. Integral broadcasting is attractive in its simplicity.

Our results. We propose and analyze three adaptive broadcasting schemes, as summarized in Table 1. In Section 2, we show that a lazy integral broadcasting scheme is exactly optimal under Measure 1, yet highly inefficient under Measure 2, which makes this scheme infeasible in practice. Nonetheless this result establishes a theoretical baseline for Measure 1 against which to compare all other algorithms. Then in Section 3 we analyze an adaptive form of harmonic broadcasting (which is nonintegral) that uses at most $\ln n$ channels at any time, as in harmonic broadcasting, but whose average number of channels used over the course of a movie is $\ln \min\{v, n\} + 1$ plus lower-order terms (Section 3). The latter bound matches, up to lower-order terms, the optimal bandwidth usage by the lazy algorithm, while providing much better performance under Measure 2. However, $\ln n$ channels is suboptimal, and in Section 4, we show that an integral adaptive pyramid broadcasting scheme is optimal under Measure 2 up to lower-order terms while still being optimal up to a constant multiplicative factor of $\lg e \approx 1.4427$ under Measure 1. Lastly in Section 5 we show that a natural greedy strategy is suboptimal for Measure 3, and furthermore that no online strategy matches the offline optimal performance when multiple movies are involved.

Table 1. Comparison of our results and harmonic broadcasting.

Broadcasting alg.	Integral?	Total bandwidth usage	Max. bandwidth usage
Harmonic [JT97, ...]	No	$\text{OPT}(n) \sim n \ln n + \gamma n$	$\ln n + \gamma$
Lazy [§2]	Yes	$\text{OPT}(n, v) \sim n \ln \min\{v, n\} + (2\gamma - 1)n$	$\sim n^{\ln 2 / \ln \ln n}$
Adapt. harmonic [§3]	No	$m \ln \min\{(n+1)v/m, n+1\} + m + v$	$\ln n + \gamma$
Adapt. pyramid [§4]	Yes	$m \lg \min\{(n+1)v/m, n+1\} + O(m)$	$\min\{v(t), \lg n\} + O(1)$

2 Lazy Broadcasting

First we consider Measure 1 in which the objective is to minimize the total amount of data transmitted (e.g., because we pay for each byte transferred). Here the goal is to maximize re-use among multiple (offset) transmissions of the same

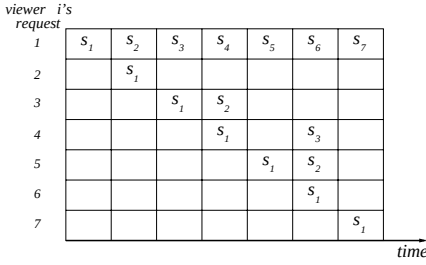


Fig. 3. Lazy broadcasting schedule with $v = n$ viewers (one every segment).

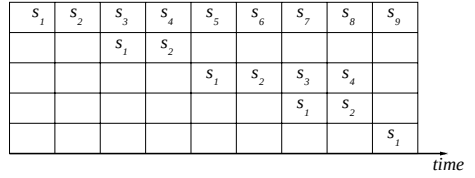


Fig. 4. Lazy broadcasting schedule adapting to $v = n/2$ viewers (one every second segment).

movie.¹ For this case, we propose *lazy broadcasting* and show that it is exactly optimal under this measure. This algorithm has a high worst-case bandwidth requirement, and hence is impractical. However, it provides the optimal baseline against which to compare all other algorithms. In particular, we show that adaptive harmonic broadcasting is within lower-order terms of lazy, and adaptive pyramid broadcasting is within a constant factor of lazy, and therefore both are also roughly optimal in terms of total data transmitted. Because the worst-case bandwidth consumption of harmonic and pyramid broadcasting is much better than that of lazy broadcasting, these algorithms will serve as effective compromises between worst-case and total bandwidth usage. The lazy algorithm sends each segment of the movie as late as possible, the moment it is required by one or more viewers; see Figure 3. All transmissions proceed at a rate of 1 (real time / play speed).

Theorem 1. *The total amount of data transmitted by the lazy algorithm is the minimum possible.*

Proof. Consider any sequence of viewers' arrival times, and a schedule A which satisfies requests of these viewers. Perform the following two operations on movie segments sent by schedule A , thereby changing A . For each time that a movie segment is sent by A :

1. If the movie segment is not required by A because every viewer can otherwise record the movie segment beforehand, delete the movie segment.
2. If the movie segment is not required when A transmits it but is required at a later time, delay sending the segment until the earliest time at which it is required and then send at full rate.

¹ Amortization of data transfers has been observed empirically over internet service provider connections. In this case, it is not uncommon to sell up to twice as much capacity than physically possible over a given link, based on the observed tendency that it is extremely uncommon for all users to reach their peak transmission rate simultaneously.

After processing all movie segments, repeat until neither operation can be done to any movie segment. This process is finite because the number of segments and time intervals during which they can be shown is finite, and each operation makes each segment nonexistent or later. The claim is that the resulting schedule is the same as the lazy schedule. The proof can be carried out by induction on the number of segments requested. $\square$

Now that we know that the lazy schedule optimizes the total amount of data transmitted, we give analytic bounds on this amount. First we study a full broadcast schedule (nonadaptive). This is equivalent to the setting in which a new viewer arrives at every time boundary between segments, see Figure 3. Notice that the i th segment is sent at time i to satisfy the request of the first viewer. The other first $i - 1$ viewers also see this transmission and record the segment. On the other hand, the $(i+1)$ st viewer did not witness this transmission, and requests the segment i time units after it started watching the movie, i.e., after time i . Hence S_i must be resent at time $2i$. In general, the i th segment must be sent at precisely those times that are multiples of i . Thus the i th segment is sent a $1/i$ fraction of the time, which shows that the total amount of bandwidth required is $n(H_n + O(1))$ for a timespan of n segments. In fact, we can obtain a more precise lower bound by observing that, at time i , we transmit those segments whose index divides i , and hence the total amount of bandwidth required is the sum of the divisors of i for $i = 1, 2, \dots, n$.

Theorem 2. *The total amount of data transmitted by the lazy algorithm for n viewers arriving at equally spaced times during a timespan of n segments for a movie n segments long is $n \ln n + (2\gamma - 1)n + O(\sqrt{n})$ segments.*

The lazy algorithm is similar to the harmonic broadcasting algorithm, the only difference being that harmonic broadcasting transmits the i th segment of the movie evenly over each period of i minutes, whereas the lazy algorithm sends it in the last minute of the interval. Comparing the bound of Theorem 2 with the total bandwidth usage of harmonic broadcasting, $n \ln n + \gamma n + O(1)$ segments, we find a difference of $\approx 0.4228n + o(n)$. Thus, harmonic broadcasting is nearly optimal under the total bandwidth metric, for $v = n$ viewers.

In contrast to harmonic broadcasting, which uses $H_n \sim \ln n$ channels at once, the worst-case bandwidth requirements of the lazy algorithm can be substantially larger:

Theorem 3. *The worst-case momentary bandwidth consumption for lazy transmission of a movie with n segments and n viewers is, asymptotically, at least $n^{\ln 2 / \ln \ln n}$.*

In the case of $v < n$ viewers (see Figure 4), Theorem 1 still shows optimality, but the bounds in Theorem 2 become weak. The next theorem gives a lower bound on the bandwidth consumed by the lazy algorithm. A matching upper bound seems difficult to prove directly, so instead we rely on upper bounds for other algorithms which match up to lower-order terms, implying that lazy is at least as good (being optimal).

Theorem 4. *The total amount of data transmitted by the lazy algorithm for v viewers arriving at equally spaced times during a timespan of n segments for a movie n segments long is at least $n \ln \min\{v, n\} + (2\gamma - 1)n + O(\sqrt{n})$ in the worst case.*

3 Adaptive Harmonic Broadcasting

In this section we propose a variation of harmonic broadcasting, called *adaptive harmonic broadcasting*, that simultaneously optimizes total bandwidth (within a lower-order term) and worst-case bandwidth usage at any moment in time. The key difference with our approach is that it adapts to a variable number of viewers over time, $v(t)$. In contrast, harmonic broadcasting is optimal only when viewers constantly arrive at every of the n movie segments.

Adaptive harmonic broadcasting defines virtual channels as in normal harmonic broadcasting, but not all virtual channels will be broadcasting at all times, saving on bandwidth. Whenever a viewing request arrives, we set the global variable t_{request} to the current time, and turn on all virtual channels. If a channel C_i was silent just before turning it on, it starts broadcasting S_i from the beginning; otherwise, the channel continues broadcasting S_i from its current position, and later returns to broadcasting the beginning of S_i . Finally, and most importantly, channel C_i stops broadcasting if the current time ever becomes larger than $t_{\text{request}} + i t_{\text{wait}}$.

Figure 2 illustrates this scheme with viewers arriving at times $t = 0, 4, 6$.

Theorem 5. *The adaptive harmonic broadcasting schedule broadcasts a movie n segments long to v active viewers with a maximum waiting time t_{wait} using at most H_n channels at any time and with a total data transfer of $m \min\{\ln(n + 1) - \ln(m/v), \ln(n + 1)\} + m + v$ segments during a timespan of $T = m t_{\text{wait}}$ minutes.*

Proof. Let t_i denote the time at which viewer i arrives, where $t_1 \leq t_2 \leq \dots \leq t_v$ and $t_v - t_1 \leq T$. Let $g_i = (t_{i+1} - t_i)/t_{\text{wait}}$ denote the normalized gaps of time between consecutive viewer arrivals.

To simplify the analysis, we do the following discretization trick. Process viewers that arrived on $[0, T]$ interval from left to right. Delete all viewers that arrived within time t_{wait} from the last viewer we considered (t_{last}) and replace them by one viewer with arrival time $t_{\text{last}} + t_{\text{wait}}$. Clearly, this only can increase bandwidth requirements in both intervals $[t_{\text{last}}, t_{\text{last}} + t_{\text{wait}}]$ and $[t_{\text{last}} + t_{\text{wait}}, T]$. Choose the next viewer that have not been considered so far and repeat the procedure. This discretizing procedure gives a method of counting “distinct” viewers, namely, on the interval $[0, T]$ there are no more than m of them and $g_i \geq 1$ for all i . In particular, if the number of viewers is more than m then adaptive harmonic broadcasting scheme degrades to simple harmonic broadcasting and theorem holds. Consider the case $v \leq m$.

The total amount of bandwidth used by all viewers is the sum over all i of the bandwidth $B(i)$ recorded by viewer i in the time interval between t_i

and t_{i+1} . Each $B(i)$ can be computed locally because we reset the clock t_{request} at every time t_i that a new viewer arrives. Specifically, $B(i)$ can be divided into (1) the set of virtual channel transmissions that completed by time t_{i+1} , that is, transmissions of length at most g_i ; and (2) the set of virtual channel transmissions that were not yet finished by time t_{i+1} , whose cost is trimmed. Each channel C_j of the first type ($j \leq g_i$) was able to transmit the entire segment in the time interval of length g_i , for a cost of one segment. Each channel C_j of the second type ($j \geq g_i$) was able to transmit for time g_i at a rate of $1/j$, for a cost of a g_i/j fraction of a segment. Thus, $B(i)$ is given by the formula

$$B(i) = \sum_{j=1}^{\lfloor g_i \rfloor} 1 + \sum_{j=\lfloor g_i \rfloor+1}^n g_i/j = \lfloor g_i \rfloor + g_i \cdot (H_n - H_{\lfloor g_i \rfloor}).$$

We have the bound $B(i) \leq g_i(1 + \ln((n+1)/g_i)) + 1$, proof omitted. Now the total amount of data transmitted can be computed by summing over all i , which gives

$$B = \sum_{i=1}^v B(i) \leq \sum_{i=1}^v \left[g_i \left(1 + \ln \frac{n+1}{g_i} \right) + 1 \right] \leq m + v + \sum_{i=1}^v g_i \ln \frac{n+1}{g_i}.$$

The last summation can be rewritten as $n+1$ times $\sum_{i=1}^v (g_i/(n+1)) \ln((n+1)/g_i)$. This expression is the entropy H , which is maximized when $g_1 = g_2 = \dots = g_v = m/v$. Hence the total amount of bandwidth B is at most $m + v + m \ln(v(n+1)/m)$, as desired when $v \leq m$. $\square$

This proof in fact establishes a tighter bound on the number of channels required, namely, the base- e entropy of the request sequence. Sequences with low entropy require less bandwidth.

4 Adaptive Pyramid Broadcasting

In this section we propose an integral adaptive broadcasting scheme which is optimal up to constant factors for both Measure 1 and Measure 2, that is, total amount of data transmitted and minimizing the maximum number of channels in use at any time.

Viswanathan and Imielinski [VI96] proposed the family of *pyramid broadcasting* schemes in which the movie is split into *chunks* of geometrically increasing size, that is, $|S_i| = \alpha|S_{i-1}|$ for some $\alpha \geq 1$, with each segment being broadcast at rate 1 using an entire physical channel.

In our case, we select $\alpha = 2$ and $|S_0| = t_{\text{wait}}$. Thus, there are $N = \lceil \lg(n+1) \rceil$ chunks $S_0, \dots, S_{N-1}$, each consisting of an integral number of segments. Chunk S_i has length $2^i t_{\text{wait}}$ (except for the last one), and hence covers the interval $[(2^i - 1)t_{\text{wait}}, (2^{i+1} - 1)t_{\text{wait}}]$ of the movie. We first analyze the bandwidth used by any protocol satisfying two natural conditions.

Lemma 1. *Consider any broadcast protocol for sending segments $S_0, \dots, S_{N-1}$ satisfying the following two conditions: (A) for every viewer, every segment is sent at most once completely, (B) no two parts of the same segment are sent in parallel; then the total bandwidth usage for v viewers within a timespan of $T = m t_{\text{wait}}$ minutes is at most $m \min\{N, N - \lg(m/v) + 1\}$ segments.*

Proof. Because there are $N = \lceil \lg(n+1) \rceil$ chunks, and no chunk is sent more than once at a time (Property B), we surely use at most $m N$ bandwidth. This bound proves the claim if $v \geq m$, so assume from now on that $v < m$.

We classify chunks into two categories. The *short* chunks are chunks S_j for $j = 0, \dots, t-1$ for some parameter t to be defined later. By Property A, chunk S_j is sent at most v times, and hence contributes at most $v 2^j$ to the bandwidth. Thus, the total bandwidth used by short chunks is at most $\sum_{j=0}^{t-1} v 2^j = v(2^t - 1)$.

The *long* chunks are chunks S_j for $j = t, \dots, N-1$. By Property B, at most one copy of S_j is sent at any given moment, so chunk S_j contributes at most m segments to the total bandwidth. Thus, the total bandwidth used by long chunks is at most $m(N-t)$. Total bandwidth $v(2^t - 1) + m(N-t)$ is minimized for the value of t roughly $\lg(m/v)$. Thus the total bandwidth is at most $m(N - \lg(m/v) + 1) - v$ segments as desired. $\square$

There are several protocols that satisfy the conditions of the previous lemma. In particular, we propose the following adaptive variant of pyramid broadcasting; see Figure 5. Suppose there are v viewers arriving at times $t_1 \leq t_2 \leq \dots \leq t_v$. We discretize the viewer arrival schedule as follows: all viewers arrived in the interval $(i t_{\text{wait}}, (i+1) t_{\text{wait}}]$ are considered as one viewer arriving at time $(i+1) t_{\text{wait}}$ for $i = 0, \dots, T/t_{\text{wait}}$ and this made-up viewer will start watching the movie immediately (i.e. at time $(i+1) t_{\text{wait}}$). Thus the waiting time for any user is at most t_{wait} , however the average waiting time is twice as less if viewers are arriving in the uniform fashion.

If at time $t_j + 2^i - 1$ viewer j has not seen the beginning of segment S_i , then this segment S_i is broadcast in full on channel i , even if viewer j has already seen parts of S_i . By this algorithm, viewer j is guaranteed to see the beginning of segment S_i by time $t_j + 2^i - 1$. Because we send segments from beginning to end, and in real time, viewer j will therefore have seen every part of S_i by the time it is needed. Furthermore, this protocol sends every segment at most once per viewer, satisfying Property A.

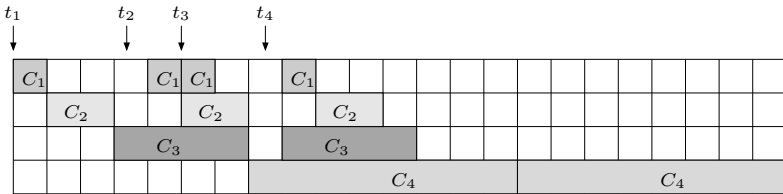


Fig. 5. Adaptive pyramid broadcasting.

It is less obvious that we never send two parts of S_i in parallel. Suppose viewer j requests segment S_i . This means that we never started broadcasting segment S_i between when j arrived (at time t_j) and when S_i is requested (at time $t_j + 2^i$). Because S_i has length 2^i , and because we send segments in their entirety, this means that at time $t_j + 2^i$ we are done with any previous transmission of S_i . Lemma 1 therefore applies, proving the first half of the following theorem:

Theorem 6. *The total bandwidth usage of adaptive pyramid broadcasting is at most $m \min\{\lceil \lg(n+1) \rceil, \lg((n+1)v/m) + 1\}$ segments, which is within a factor of $\lg e \approx 1.4427$ (plus lower order terms) of optimal. Furthermore, the maximum number of channels in use at any moment t is at most $\min\{v(t), \lceil \lg(n+1) \rceil\}$.*

Finally, we prove that the maximum number of channels used by adaptive pyramid broadcasting is optimal among all online adaptive broadcasting algorithms: in a strong sense, the number of channels cannot be smaller than v , unless v is as large as $\sim \lg n$.

Theorem 7. *Consider any online adaptive broadcasting algorithm that at time t uses $c(t)$ physical channels to serve $v(t)$ current viewers and for which $c(t) \leq v(t)$ at all times t . Then there is a sequence of requests such that, for all $v \in \{1, 2, \dots, \lg n - \lg \lg n\}$, there is a time t when $v(t) = v$ and $c(t) = v(t)$.*

Proof. Consider the sequence of requests at times $0, \frac{1}{2}n, \frac{3}{4}n, \frac{7}{8}n, \frac{15}{16}n, \dots, (1 - 1/2^i)n, \dots$. In this sequence of requests, we claim that no re-use of common segments between different viewers is possible in the time interval $[0, n)$. Consider the i th viewer, who arrives and starts recording at time $(1 - 1/2^i)n$. In the time interval $[(1 - 1/2^i)n, n)$, the i th viewer needs to be sent the first $n/2^i - 1$ segments of the movie. (The -1 term is because the viewer waits for the first time unit (segment), and only starts watching at time $(1 - 1/2^i)n + 1$.) But all previously arriving viewers must have already been sent those segments before time $(1 - 1/2^i)n$, because by construction they have already watched them by that time. Therefore, no segments that are watched by any viewer in the time interval $[0, n)$ can have their transmissions shared between viewers.

Now define the *buffer amount* of a viewer to be the amount of time that each viewer is “ahead”, i.e., the amount of time that a viewer could wait before needing its own rate-1 broadcast until the end of the movie (which is at least time n). Because there is no re-use between viewers, we maintain the invariant that, if there are v current viewers, then the total buffer amount of all viewers is at most v . A buffer amount of 1 for each viewer is easy to achieve, by having each viewer record for the one unit of wait time on its own rate-1 channel. It is also possible to “transfer” a buffer amount from one viewer to another, by partially using one viewer’s channel to send part of another viewer’s needed segment, but this operation never strictly increases the total buffer amount.

In the time interval $[(1 - 1/2^{v-1})n, (1 - 1/2^v)n)$, there are exactly v active viewers, and each viewer needs to watch $(1/2^{v-1} - 1/2^v)n$ segments, except for one viewer who watches one fewer segment. Viewers might during this time “use up” their buffer amount, by using their channels for the benefit of

other viewers, catching up to real time. However, this can only decrease the resource requirement during this time interval by up to v , so the total resource requirement is still at least $v(1/2^{v-1} - 1/2^v)n - v - 1$. On the other hand, if there are c physical channels in use during this time interval when exactly v viewers are active, then the maximum bandwidth usable in this time interval, $c(1/2^{v-1} - 1/2^v)n$, must be at least the resource requirement. Thus, $c \geq v - (v - 1)/((1/2^{v-1} - 1/2^v)n) = v - 2^v(v - 1)/n$.

Because c measures physical channels, c is integral, so the bound on c in fact implies $c \geq v$ provided the error $2^v(v - 1)/n$ is less than 1. If $v \leq \lg n - \lg \lg n$, then $2^v(v - 1) = (n/\lg n)(\lg n - \lg \lg n - 1) < n$. Therefore, for any v in this range (as claimed in the theorem), we need as many physical channels as viewers. $\square$

Adaptive pyramid broadcasting inherits the simplicity-of-implementation properties that have made pyramid broadcasting popular: not only is the algorithm integral on segments, it is integral on chunks, always broadcasting entire segments from beginning to end in real time.

5 Greedy Broadcasting and Offline Scheduling

Suppose we have a fixed amount of available bandwidth, and our goal is to satisfy as many viewers as possible. The natural greedy algorithm is to send the segments of a movie that are required soonest, as soon as there is available bandwidth. We imagine a *wavefront* sweeping through the requests in the order that they are needed. The *front time* must always remain ahead of real time, or in the worst case equal to real time. If the front time ever falls behind real time, some viewer will not be satisfied. The greedy algorithm is suboptimal in the following sense:

Theorem 8. *There is a sequence of requests for a single movie that is satisfiable within a fixed available bandwidth but for which the greedy algorithm fails to find a satisfactory broadcast schedule.*

Theorem 9. *There is a family of request sequences for two movies that is satisfiable offline within a fixed available bandwidth, but which can force any online scheduling algorithm to fail.*

6 Conclusions and Open Questions

We introduced the concept of adaptive broadcasting schedules which gracefully adjust to varying numbers of viewers. We measured the performance of three new algorithms under two metrics inspired by realistic bandwidth cost considerations. In particular, we showed that adaptive harmonic broadcasting is optimal up to lower-order terms under total amount of data transmitted, and that adaptive pyramid broadcasting achieves optimal maximum channel use at the cost of a constant factor penalty on the total amount of data transmitted. All the algorithms generalize to multiple different-length movies being watched by different numbers of viewers, and the same worst-case optimality results carry over.

We also showed that any online algorithm might fail to satisfy a given bandwidth requirement that is satisfiable offline for a two-movie schedule. One open question is to determine the best competitive ratio on the fixed bandwidth bound achievable by online broadcasting schedules versus the offline optimal.

References

- [AB+96] A. Albanese, J. Blörnet, J. Edmonds, M. Luby and M. Sudan. Priority encoding transmission. *IEEE Trans. Inform. Theory*, 42(6):1737–1744, 1996.
- [B-NL02] A. Bar-Noy and R. E. Ladner. Windows scheduling problems for broadcast systems. *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 433–442, 2002.
- [CP01] S. R. Carter, J.-F. Paris, S. Mohan and D. D. E. Long. A dynamic heuristic broadcasting protocol for video-on-demand. *Proc. 21st International Conference on Distributed Computing Systems*, pages 657–664, 2001.
- [DSS94] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems*, 4(3):112–121, 1996.
- [ES02] L. Engebretsen and M. Sudan. Harmonic broadcasting is bandwidth-optimal assuming constant bit rate *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 431–432, 2002.
- [EVZ00] D. L. Eager, M. K. Vernon and J. Zahorjan. Bandwidth skimming: a technique for cost-effective video-on-demand. *Proc. IS&T/SPIE Conference on Multimedia Computing and Networking (MMCN)*, pages 206–215, 2000.
- [EVZ01] D. L. Eager, M. K. Vernon and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Transactions on Knowledge and Data Engineering*, 3(5):742–757, 2001.
- [JT97] L. Juhn and L. Tseng. Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271, 1997.
- [JT98] L. Juhn and L. Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Trans. on Broadcasting*, 44(1):100–105, 1998.
- [ME+01] A. Mahanti, D. L. Eager, M. K. Vernon and D. Sundaram-Stukel. Scalable on-demand media streaming with packet loss recovery. *Proc. 2001 ACM Conf. on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM'01)*, pp. 97–108, 2001.
- [PCL98a] J.-F. Pâris, S. W. Carter and D. D. E. Long. Efficient broadcasting protocols for video on demand. *Proc. 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 127–132, 1998.
- [PCL98b] J.-F. Pâris, S. W. Carter and D. D. E. Long. A low bandwidth broadcasting protocol for video on demand. *Proc. 7th International Conference on Computer Communications and Networks*, pages 690–697, 1998.
- [VI96] S. Viswanathan and T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197–208, 1996.
- [Won88] J. W. Wong. Broadcast delivery. *Proc. of the IEEE*, 76(12):1566–1577, 1988.

Multi-player and Multi-round Auctions with Severely Bounded Communication

Liad Blumrosen¹, Noam Nisan¹, and Ilya Segal²

¹ School of Engineering and Computer Science.
The Hebrew University of Jerusalem, Jerusalem, Israel.

`{liad,noam}@cs.huji.ac.il`

² Department of Economics,
Stanford University, Stanford, CA 94305
`ilya.segal@stanford.edu`

Abstract. We study auctions in which bidders have severe constraints on the size of messages they are allowed to send to the auctioneer. In such auctions, each bidder has a set of k possible bids (i.e. he can send up to $t = \log(k)$ bits to the mechanism). This paper studies the loss of economic efficiency and revenue in such mechanisms, compared with the case of unconstrained communication. For any number of players, we present auctions that incur an efficiency loss and a revenue loss of $O(\frac{1}{k^2})$, and we show that this upper bound is tight. When we allow the players to send their bits sequentially, we can construct even more efficient mechanisms, but only up to a factor of 2 in the amount of communication needed. We also show that when the players' valuations for the item are not independently distributed, we cannot do much better than a trivial mechanism.

1 Introduction

Computers on the Internet are owned by different parties with individual preferences. Trying to impose protocols and algorithms on them in the traditional computer-science way is doomed to fail, since each party might act for its own selfish benefit. Thus, designing protocols for Internet-like environments requires the usage of tools from other disciplines, especially microeconomic theory and game theory. This intersection between computer science theory and economic theory raises many interesting questions. Indeed, much theoretical attention was given in recent years to problems with both game theoretic and algorithmic aspects (see e.g. the surveys [10,18,5]). Many of the algorithms for such distributed environments are closely related to the theory of *mechanism design* and in particular to *auction theory* (see [6] for comprehensive survey about auctions). An auction is actually an algorithm, that allocates some resources among a set of players. The messages (bids) that the players send to the auctioneer are the input for this algorithm, and it outputs an allocation of the resources and payments for the players. The main challenge in designing auctions is related to the incomplete information that the designer has about the players' secret data (for example,

how much they are willing to pay for a certain resource). The auction mechanism must somehow elicit this information from the selfish participants, in order to achieve global or “social” goals (e.g. maximize the seller’s revenue). Recent results show that auctions are hard to implement in practice. The reasons might be computational (see e.g. [12,8]), communication-related ([13]), uncertainty about timing or participants ([4,7].) and many more. This, and the growing usage of auctions in e-commerce (e.g. [14,21,15]) and in various computing systems (see e.g. [11,19,20]) led researchers to take computational effects into consideration when designing auctions.

Much interest was given in the economic literature to the design of *optimal auctions* and *efficient auctions*. Optimal auctions are auctions that maximize the seller’s revenue. Efficient auctions maximize the social welfare, i.e. they allocate the resources to the players that want them the most. A positive correlation usually exist between the two measures: a player is willing to pay more for an item that is worth a higher value to her. Nevertheless, efficient auctions are not necessarily optimal, and vice versa. In our model, each player has a private valuation for a single item (i.e. she knows how much she values the item, but this value is a private information for herself). The goal of the auction’s designer (in the Bayesian framework) is, given distributions on the players’ valuations, to find auctions that maximize the expected revenue or the expected welfare, when the players act selfishly. For the single item case, these problems are in fact solved: the Vickrey auction (or the 2nd-price auction, see [17]) is efficient; Myerson, in a classic paper ([9]), fully characterize optimal auctions when the players’ valuations are independently distributed. In the same paper, Myerson also shows that Vickrey’s auction (with some reservation price) is also optimal (i.e. revenue maximizing), when the distribution functions hold some regularity property. Optimal auctions and efficient auctions were studied lately also by computer scientists (e.g. [4,16]). Recently, Blumrosen and Nisan ([1]) initiated the study of auctions with severely bounded communication, i.e. settings where each player can send a message of up to t bits to the mechanism. In other words, each bidder can choose a bid out of a set of $k = 2^t$ possible bids. The players’ valuations, however, can be any real numbers in the range $[0, 1]$.

Here, we generalize the main results from [1] for multi-player games. We also study the effect of relaxing some of the assumptions made in [1], namely the simultaneous bidding and the independence of the valuations.

Severe constraints on the communication are expected in settings where we need to design quick, and cheap auctions that should be performed frequently. For example, if a route for a packet over the Internet is auctioned, we can dedicate for this purpose only a small number of bits. Otherwise, the network will be congested very quickly. For example, we might want to use some unused bits in existing networking protocols (e.g. IP or TCP) to transfer the bidding information. This is opposed to the traditional economic approach that views the information sent by the players as real numbers (representing these can take infinite number of bits!). Low communication also serves as a proxy for other desirable properties: with low communication the interface for the auction is

simpler (the players have a small number of possible bids to choose from), the information revelation is smaller and only a small number of discrete prices is used. In addition, understanding the tradeoffs between communication and auctions' optimality (or efficiency) might help us find feasible solutions for settings which are currently computationally impossible (combinatorial auctions' design is the most prominent example).

Under severe communication restrictions, [1] characterizes optimal and efficient auctions among *two players*. They prove that the welfare loss and the revenue loss in mechanisms with t -bits messages is mild: for example, with only *one* bit allowed for each player (i.e. $t = 1$) we can have 97 percent of the efficiency achieved by auctions that allow the players to send infinite number of bits (with uniform distributions)! Asymptotically, they show that the loss (for both measures) diminishes exponentially in t (specifically $O(\frac{1}{2^{2t}})$ or $O(\frac{1}{k^2})$ where $k = 2^t$). These upper bounds are tight: for particular distribution functions, the expected welfare loss and the expected revenue loss in *any* mechanism are $\Omega(\frac{1}{k^2})$.

In this work, we show n -player mechanisms that, despite using very low communication, are nearly optimal (or nearly efficient). These mechanisms are an extension of the "*priority-games*" and "*modified priority-games*" concepts described in [1], and they achieve the asymptotically-optimal results with dominant strategies equilibrium and with individual-rationality constraints (see formal definitions in the body of the paper). For both measures, we characterize mechanisms that incur a loss of $O(\frac{1}{k^2})$, and we show that for some distribution functions (e.g. the uniform distribution) this bound is tight.

We also extend the framework to the following settings:

- **Multi-round auctions:** By allowing the bidders to send the bits of their messages one bit at a time, in alternating order, we can strictly increase the efficiency of auctions with bounded communication. In such auctions, each player knows what bits were sent by all players up to each stage. However, we show that the same extra gain can be achieved in simultaneous auctions that use less than double amount of communication.
- **Joint distributions:** When the players' valuations are statistically dependent, we show that we cannot do better (asymptotically) than a trivial mechanism that achieves an efficiency loss of $O(\frac{1}{k})$. Specifically, we show that for some joint distribution functions, *every* mechanism with k possible bids incurs a revenue loss of at least $\Omega(\frac{1}{k})$.
- **Bounded distribution functions:** We know ([1]) that we cannot construct one mechanism that incurs a welfare loss of $O(\frac{1}{k^2})$ for **all** distribution functions. Nevertheless, if we assume that the density functions are bounded from above or from below, a trivial mechanism achieves results which are asymptotically optimal.

The organization of the paper is as follows: section 2 describes the formal model of auctions with bounded communication. Section 3 gives tight upper bounds for the optimal welfare loss and revenue loss in n -player mechanisms. Section 4 studies the case of bounded density functions and joint distributions.

B	0	1
A		
0	B wins and pays 0	B wins and pays 0
1	A wins and pays $\frac{1}{3}$	B wins and pays $\frac{2}{3}$

Fig. 1. A matrix representation for a mechanism with two possible bids. E.g., when Alice bids "1" and Bob bids "0", Alice wins the item and pays $\frac{1}{3}$.

Finally, section 5 discusses multi round auctions. All the omitted proofs can be found in the full version ([2]).

2 The Model

We consider single item, sealed bid auctions among n risk-neutral players. Player i has a private data (*valuation*) $v_i \in [0, 1]$ that represents the maximal payment he is willing to pay for the item. For every player i , v_i is independently drawn from a density function f_i ($\int_0^1 f_i(v)dv = 1$) which is commonly known for all participants. The cumulative distribution for player i is F_i . Throughout the paper we assume that the distribution functions are continuous and always positive. We also assume a normalized model, i.e. players' valuations for not having the item are zero. The seller's valuation for the item is zero, and the players' valuations depend only on whether they win the item or not (no externalities).

Players aim to maximize their utilities, which are *quasi-linear*, i.e. the utility of player i from the item is $v_i - p_i$ when p_i is his payment.

The unique assumption in our model, is that each player can send a message of no more than $t = \lg(k)$ **bits** to the mechanism, i.e. players can choose one of k possible **bids** (or messages). Denote the possible set of bids for the players as $\beta = \{0, 1, 2, \dots, k-1\}$. In each auction, player i chooses a bid $b_i \in \beta$. A mechanism determines the allocation and payments given a vector of bids $b = (b_1, \dots, b_n)$:

Definition 1 A mechanism g is composed of a pair (a, p) where:

- $a : (\beta \times \dots \times \beta) \rightarrow [0, 1]^n$ is the allocation scheme. We denote the i 'th coordinate of $a(b)$ by $a_i(b)$, which is player i 's probability for winning the item when the bidders bid b . Clearly, $\forall i \forall b a_i(b) \geq 0$ and $\forall b \sum_{i=1}^n a_i(b) \leq 1$.
- $p : (\beta \times \dots \times \beta) \rightarrow \mathbb{R}^n$ is the payment scheme. $p_i(b)$ is player i 's payment given a bids' vector b (paid only upon winning).

Definition 2 In a mechanism with k -possible bids, $|\beta| = k$. Denote the set of all mechanisms with k -possible bids among n players by $\mathbf{G}_{n,k}$.

Figure 1 describes the matrix representation of a 2-player mechanism with two possible bids ("0" or "1").

All the results in this paper are achieved with *ex-post Individually-Rational* (IR) mechanisms, i.e. mechanisms in which players can always ensure themselves not to pay more than their valuations for the item (or 0 when they lose). (We equivalently use the term: mechanisms with ex-post individual rationality.)

Definition 3 A strategy s_i for player i in a game $g \in G_{n,k}$ describes how the player determines his bid according to his valuation, i.e. it is a function $s_i : [0, 1] \rightarrow \{0, 1, \dots, k-1\}$.

Denote $\varphi_k = \{s \mid s : [0, 1] \rightarrow \{0, 1, \dots, k-1\}\}$ (i.e. the set of all strategies for players with k possible bids).

Definition 4 A real vector $c = (c_0, c_1, \dots, c_k)$ is a vector of threshold-values if $c_0 \leq c_1 \leq \dots \leq c_k$.

Definition 5 A strategy $s_i \in \varphi_k$ is a **threshold-strategy based on a vector of threshold-values** $c = (c_0, c_1, \dots, c_k)$, if $c_0 = 0$ and $c_k = 1$ and for every $c_i \leq v_i < c_{i+1}$ we have $s_i(v_i) = i$. We say that s_i is a **threshold strategy**, if there exists a vector c of threshold values such that s_i is a threshold strategy based on c .

We use the notations: $s(v) = (s_1(v_1), \dots, s_n(v_n))$, when s_i is a strategy for bidder i and $v = (v_1, \dots, v_n)$. Let s_{-i} denote the strategies of the players except i , i.e. $s_{-i} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$. We sometimes use the notation $s = (s_i, s_{-i})$.

2.1 Optimality Measures

The players in our model choose strategies that maximize their utilities. We are interested in games with stable behaviour for all players, i.e. such that these strategies form an equilibrium.

Definition 6 Let $u_i(g, s)$ be the expected utility of player i from game g when bidders use the strategies s , i.e. $u_i(g, s) = E_{v \in [0,1]^n} (a_i(s(v)) \cdot (v_i - p_i(s(v))))$

Definition 7 The strategies $s = (s_1, \dots, s_n)$ form a **Bayesian-Nash equilibrium** in a mechanism $g \in G_{n,k}$, if for every player i , s_i is the best response for the strategies s_{-i} of the other players, i.e. $\forall i \quad \forall \tilde{s}_i \in \varphi_k \quad u_i(g, (s_i, s_{-i})) \geq u_i(g, (\tilde{s}_i, s_{-i}))$

Definition 8 A strategy s_i for player i is **dominant** in mechanism $g \in G_{n,k}$ if regardless of the other players' strategies s_{-i} , i cannot gain a higher utility by changing his strategy, i.e. $\forall \tilde{s}_i \in \varphi_k \quad \forall s_{-i} \quad u_i(g, (s_i, s_{-i})) \geq u_i(g, (\tilde{s}_i, s_{-i}))$

We say that a mechanism g has a **dominant strategies equilibrium** if for every player i there exists a strategy s_i which is dominant. Clearly, a dominant strategies equilibrium is also a Bayesian-Nash equilibrium.

Each bidder aims to maximize her expected utility. As mechanisms' designers, we aim to optimize "social" criteria such as *welfare* (efficiency) and *revenue*. The *expected welfare* from a mechanism g , when bidders use strategies s , is the expected valuation of the winning players (if any).

Definition 9 Let $w(g, s)$ denote the expected welfare in the n -player game g when bidders' strategies are s , i.e. $w(g, s) = E_{v \in [0,1]^n} (\sum_{i=1}^n a_i(s(v)) \cdot v_i)$

Definition 10 Let $r(g, s)$ denote the expected revenue in the n -player game g when bidders' strategies are s , i.e. $r(g, s) = E_{v \in [0,1]^n} (\sum_{i=1}^n a_i(s(v)) \cdot p_i(s(v)))$

Definition 11 We say that a mechanism $g \in G_{n,k}$ achieves an expected welfare (**revenue**) of α if g has a Bayesian-Nash equilibrium s for which the expected welfare (**revenue**) is α , i.e. $w(g, s) = \alpha$ ($r(g, s) = \alpha$).

Definition 12 We say that a mechanism $g \in G_{n,k}$ incurs a welfare loss of c , if there is a Bayesian-Nash equilibrium s in g such that the difference between $w(g, s)$ and the maximal welfare with unbounded communication is c .

We say that g incurs a revenue loss of c , if there is an individually-rational Bayesian-Nash equilibrium s in g , such that the difference between $r(g, s)$ and the optimal revenue, achieved in an individually-rational mechanism with Bayesian-Nash equilibrium in the unbounded communication case, is c .

Recall that an equilibrium is individually rational, if the expected utility of each player, given his own valuation, is non negative. The mechanism described in Fig. 1 has a dominant strategy equilibrium that achieves an expected welfare of $\frac{35}{54}$ (with uniform distributions). Alice's dominant strategy is the threshold strategy based on $\frac{1}{3}$, i.e. she bids "0" when her valuation is below $\frac{1}{3}$, and "1" otherwise. The threshold strategy based on $\frac{2}{3}$ is dominant for Bob. We know ([17]) that the optimal welfare from a 2-player auction with unconstrained communication is $\frac{2}{3}$. Thus, the welfare loss incurred by this mechanism is $\frac{2}{3} - \frac{35}{54} = \frac{1}{54}$.

3 Multi-player Mechanisms

In this section, we construct n -player mechanisms with bounded communication which are asymptotically optimal (or efficient). We prove that they incur losses of welfare and revenue of $O(\frac{1}{k^2})$, and that these upper bounds are tight.

It was shown in [1] that "priority-games" (PG) and "modified priority-games" (MPG) are efficient and optimal (respectively) among all the 2-player mechanisms with bounded communications. For the n -player case, the characterization of the welfare maximizing and the revenue maximizing mechanisms remains an open question. We conjecture that PG's (and MPG's) with optimally chosen payments are efficient (optimal). We show that PG's and MPG's achieve **asymptotically**-optimal welfare and revenue (respectively). Note, that even though our model allows lotteries, our analysis presents only deterministic mechanisms. Indeed, [1] shows that optimal results are achieved by deterministic mechanisms.

Definition 13 A game is called a **priority-game** if it allocates the item to the player i that bids the highest bid (i.e. when $b_i > b_j$ for all $j \neq i$, the allocation is $a_i(b) = 1$ and $a_j(b) = 0$ for $j \neq i$), with ties consistently broken according to a pre-defined order on the players.

For example, Fig. 1 describes a priority game: the player with the highest bid wins, and ties are always broken in favour of Bob.

Definition 14 A game is called a **modified priority-game** if it has an allocation as in priority-games, but no allocation is done when all players bid 0.

Definition 15 An n -player priority-game based on a profile of threshold values' vectors $\vec{t} = (t^1, \dots, t^n) \in \times_{i=1}^n \mathbb{R}^{k+1}$ (where for every i , $t_0^i \leq t_1^i \leq \dots \leq t_k^i$) is a mechanism that its allocation is as in a priority game and its payment scheme is as follows: when player j wins the item for the bids vector b she pays the smallest valuation she might have and still win the item, given that she uses the threshold strategy s_j based on t^j . I.e. $p_j(b) = \min\{v_j | a_j(s_j(v_j), b_{-j}) = 1\}$. We denote this mechanism as $PG_k(\vec{t})$. A modified priority game with a similar payment rule is called a modified priority-game based on a profile of threshold values' vectors, and is denoted by $MPG_k(\vec{t})$.

For example, Fig. 1 describes a priority game based on the threshold values $(0, \frac{1}{3}, 1)$ and $(0, \frac{2}{3}, 1)$. When Bob bids 0, the minimal valuation of Alice for which she still wins is $\frac{1}{3}$, thus this is her payment upon winning, and so on. We first show that these mechanisms have dominant-strategies and ex-post IR:

Proposition 1 For every profile of identical threshold values' vectors $\vec{t} = (x, x, \dots, x)$, $x \in \mathbb{R}^{k+1}$ and $x_0 \leq x_1 \leq \dots \leq x_k$, the threshold-strategies based on these threshold values are dominant in $PG_k(\vec{t})$, and this mechanism is ex-post IR.

3.1 Asymptotically Efficient Mechanisms

Now, we show that given any set of n distribution functions of the players, we can construct a mechanism that incurs a welfare loss of $O(\frac{1}{k^2})$. In [1], a similar upper bound was given for the case of 2-player mechanisms:

Theorem 1 [1] For every set of distribution functions on the players' valuations, the 2 player mechanism $PG_k(x, y)$ incurs an expected welfare loss of $O(\frac{1}{k^2})$ (for some threshold values vectors x, y). Moreover, when all valuations are distributed uniformly, the expected welfare loss is at least $\Omega(\frac{1}{k^2})$ in any mechanism.

Here, we prove that n -player priority games are asymptotically efficient:

Theorem 2 For any number of players n , and for any set of distribution functions of the players' valuations, the mechanism $PG_k(\vec{t})$ incurs a welfare loss of $O(\frac{1}{k^2})$, for some threshold values vector $\vec{t} \in \times_{i=1}^n \mathbb{R}^{k+1}$. This mechanism has a dominant-strategies equilibrium with ex-post IR.

In the following theorem we show that for uniform distributions, the welfare loss is proportional to $\frac{1}{k^2}$:

Theorem 3 When valuations are distributed uniformly, and for any (fixed) number of players n , any mechanism $g \in G_{n,k}$ incurs a welfare loss of $\Omega(\frac{1}{k^2})$.

Proof. Consider only the case where players 1 and 2 have valuations greater than $\frac{1}{2}$, and the rest of the players have valuations below $\frac{1}{2}$. This occurs with the constant probability of $\frac{1}{2^n}$ (n is fixed). For maximal efficiency, a mechanism with k possible bids always allocates the item to player 1 or 2. But due to theorem 1, a welfare loss of $\Omega(\frac{1}{k^2})$ will still be incurred (the fact that in theorem 1 the valuations' range is $[0, 1]$ and here it is $[\frac{1}{2}, 1]$ only changes the constant c). Thus, any mechanism will incur a welfare loss which is $\Omega(\frac{1}{k^2})$.

3.2 Asymptotically Optimal Mechanisms

Now, we present mechanisms that achieve asymptotically optimal expected *revenue*. We show how to construct such mechanisms and give tight upper bounds for the revenue loss they incur.

Most results in the economic literature on revenue-maximizing auctions, assume that the distribution functions of the players' valuations holds a *regularity* property (as defined by Myerson [9], see below). For example, only when the valuations of all players are distributed with the same regular distribution-function, it is known that Vickrey's 2nd-price auction, with an appropriately chosen reservation price, is revenue-optimal ([17,9,3]).

Definition 16 ([9]) *Let f be a density function, and let F be its cumulative function. We say that f is regular, if the function $\tilde{v}(\mathbf{v}) = \mathbf{v} - \frac{1-F(\mathbf{v})}{f(\mathbf{v})}$ is monotone, strictly increasing function of v . We call $\tilde{v}$ the virtual utility.*

We define the virtual utility of all the players, except the winner, as zero. The seller's virtual utility is equal to his valuation for the item (zero in our model). Myerson ([9]) observed that in equilibrium, the expected revenue equals the expected virtual-utility (i.e. the average virtual utility of the winning players):

Theorem 4 ([9]) *Consider a model with unbounded communication, in which losing players pay zero. Let h be a direct-revelation mechanism, which is incentive compatible (i.e. truth telling by all players forms Nash equilibrium) and individually rational. Then in h , the expected revenue equals the expected virtual utility.*

Simple arguments show (see [1]) that Myerson's observation also holds for auctions with bounded communication:

Proposition 2 ([1]) *Let $g \in G_{n,k}$ be a mechanism with Bayesian Nash equilibrium $s = (s_1, \dots, s_n)$ and ex-post individual rationality. Then, the expected revenue of s in g is equal to the expected virtual-utility in g .*

Using this property, the revenue optimization problem can be reduced to a welfare optimization problem, which was solved for the n -player case in theorems 2 and 3. We extend the techniques used in [1] for the n -player case: we optimize the expected *welfare* in settings where the players consider their virtual utility as their valuations (see [2] for the proof). We show that for a fixed n , and for

every regular distribution, there is a mechanism that incurs a revenue loss of $O(\frac{1}{k^2})$. Again, this bound is tight: for uniform distributions the optimal revenue loss is proportional to $\frac{1}{k^2}$.

Theorem 5 *Assume that all valuations are distributed with the same regular distribution function. Then, for any number of players n , $MPG_k(\vec{t})$ incurs a revenue loss of $O(\frac{1}{k^2})$, for some threshold values vector $\vec{t} \in \times_{i=1}^n \mathbb{R}^{k+1}$. This mechanism has dominant strategies equilibrium with ex-post IR.*

Theorem 6 *Assume that the players' valuations are distributed uniformly. Then, for any (fixed) number of players n , any mechanism $g \in G_{n,k}$ incurs a revenue loss of $\Omega(\frac{1}{k^2})$.*

4 Bounded Distributions and Joint Distributions

In previous theorems, we showed how to construct mechanisms with asymptotically optimal welfare and revenue, given a set of distribution functions. Can we design a particular mechanism that achieve similar results for **all** distribution functions? Due to [1], the answer in general is no. The simple mechanism $PG_k(x, x)$ where $x = (0, \frac{1}{k}, \frac{2}{k}, \dots, \frac{k-1}{k}, 1)$ incurs a welfare loss of $O(\frac{1}{k})$ and no better upper bound can be achieved. Nevertheless, we show that if the distribution functions are bounded from above or from below, this trivial mechanism for two players achieves an expected welfare which is asymptotically optimal.

Definition 17 *We say that a density function f is bounded from above (**below**) if for every x in its domain, $f(x) \leq c$ ($f(x) \geq c$), for some constant c .*

Proposition 3 *For every pair of distribution functions of the players' valuations which are bounded from above, the mechanism $PG_k(x, x)$, where $x = (0, \frac{1}{k}, \frac{2}{k}, \dots, \frac{k-1}{k}, 1)$, incurs an expected welfare loss of $O(\frac{1}{k^2})$.*

*For every pair of distribution functions which are bounded from below, **every** mechanism incurs an expected welfare loss of $\Omega(\frac{1}{k^2})$.*

So far, we assumed that the players' valuations are drawn from statistically independent distributions. Now, we relax this assumption and deal with general joint distributions of the valuations. For this case, we show that a trivial mechanism is actually the best we can do (asymptotically). Particularly, it derives a tight upper bound of $O(\frac{1}{k})$ for the efficiency loss in 2-player games.

Theorem 7 *The mechanism $PG_k(x, x)$ where $x = (0, \frac{1}{k}, \frac{2}{k}, \dots, \frac{k-1}{k}, 1)$ incurs an expected welfare loss $\leq \frac{1}{k}$ for any joint distribution ϕ on the players' valuations. Moreover, for every k there is a joint distribution function ϕ_k such that **every** mechanism $g \in G_{2,k}$ incurs a welfare loss $\geq c \cdot \frac{1}{k}$ (where c is some positive constant independent of k).*

		B	
		0	1
A	0	$A, 0$	$B, \frac{1}{4}$
	1	$A, \frac{1}{3}$	$B, \frac{3}{4}$

Fig. 2. (h_1) This sequential game (when A bids first, then B) achieves higher expected welfare than any simultaneous mechanism with the same communication complexity (2 bits). The welfare is achieved with Bayesian-Nash equilibrium.

5 Multi-round Auctions

In previous sections, we analyzed auctions with bounded communication in which players simultaneously send their bids to the mechanism. Can we get better results with *multi-round* (or *sequential*) mechanisms? I.e. mechanisms in which players send their bids one bit at a time, in alternating order. In this section, we show that sequential mechanisms can achieve better results. However, the additional gain (in the amount of communication) is up to a factor of 2.

5.1 Sequential Mechanisms Can Do Better

The definitions in this section are similar in spirit to the model described in section 2. For simplicity, we present this model less formally.

Definition 18 *A sequential (or multi-round) mechanism is a mechanism in which players send their bids one bit at a time, in alternating order. In each stage, each player knows the bits the other players sent so far. Only after all the bits were transmitted, the mechanism determines the allocation and payments.*

Definition 19 *The communication complexity of a mechanism is the total amount of bits which are sent by the players.*

Definition 20 *A strategy for a player in a sequential mechanism is the way she determines the bits she transmits, at every stage, given her valuation and given the other players' bits up to this stage.*

A strategy for a player in a sequential mechanism is called a threshold strategy if in each stage i of the game, the player determines the bit she sends according to some threshold value x_i ; I.e. if her valuation is smaller than this threshold she bids 0, or bids 1 otherwise.

Denote the following sequential mechanism by h_1 (see Fig. 2): Alice sends one bit to the mechanism first. Bob, knowing Alice's bid, also sends one bit. When Alice bids 0: Bob wins if he bids 1 and pays $\frac{1}{4}$; If he bids zero Alice wins and pays zero. When Alice bids 1: Bob also wins when he bids 1, but now he pays $\frac{3}{4}$; If he bids zero, Alice wins again, but now she pays $\frac{1}{3}$.

The communication complexity of this mechanism is 2 (each player sends one bit to the mechanism). When players' valuations are distributed uniformly, this mechanism achieves an expected welfare which is greater than the optimal welfare from simultaneous mechanisms with the same communication complexity:

Proposition 4 *When valuations are distributed uniformly, the mechanism h_1 above has a Bayesian-Nash equilibrium and an expected welfare of 0.653.*

Proof. Consider the following strategies: Alice uses a threshold strategy based on the threshold value $\frac{1}{2}$, and Bob uses the threshold $\frac{1}{4}$ when Alice bids “0” and the threshold $\frac{3}{4}$ when Alice bids 1. It is easy to see that these strategies form a Bayesian-Nash equilibrium, with expected welfare of 0.653.

The communication complexity of the mechanism h_1 above is 2 bits (each player sends one bit). The efficient simultaneous mechanism, with 2 bits’ complexity, achieves an expected welfare of 0.648 ([1]). Thus, we can gain more efficiency with sequential mechanisms. Note that this expected welfare is achieved in h_1 with Bayesian-Nash equilibrium, as opposed to dominant strategies equilibria in all previous results.

5.2 The Extra Gain from Sequential Mechanisms Is Limited

How significant is the extra gain from sequential mechanisms? The following theorem states that for every sequential mechanism there exists a simultaneous mechanism that achieves at least the same welfare with less than double amount of communication. Note that in sequential mechanisms the players must be informed about the bits the other players sent (we do not take this into account in our analysis), so the total gain in communication can be very mild. We start by proving that optimal welfare can be achieved with threshold-strategies.

Lemma 1 *Given a sequential mechanism h and a profile of strategies $s = (s_1, \dots, s_n)$ of the players, there exists a profile of **threshold** strategies $\bar{s} = (\bar{s}_1, \dots, \bar{s}_n)$ that achieves at least the same welfare with h as s does.*

Theorem 8 *Let h be a 2-player sequential mechanism with communication complexity m . Then, there exists a **simultaneous** mechanism g that achieves at least the same expected welfare as h , with communication complexity of $2m - 1$.*

Proof. Consider a 2-player, sequential mechanism h with a Bayesian-Nash equilibrium, and with communication complexity m (we assume m is even, i.e. each player sends $\frac{m}{2}$ bits). Due to lemma 1, there exists a profile $s = (s_1, s_2)$ of threshold-strategies that achieves at least the same expected welfare on h as the equilibrium welfare. Now, we will count the number of different thresholds of player A : at stage 1, she uses a single threshold. After B sends his first bit, A also uses a threshold, but she might have a different one for each history, i.e. $2^2 = 4$ thresholds. This way, it is easy to see that the number of thresholds for A is: $\alpha_A(m) = 2^0 + 2^2 + \dots + 2^{m-2}$, and for player B is $\alpha_B(m) = 2^1 + 2^3 + \dots + 2^{m-1}$. Next, we construct a simultaneous mechanism g that achieves at least the same expected welfare with a communication complexity smaller than $2m - 1$. In g , each player simply “tells” the mechanism within which 2 of the threshold values his valuations is. The number of bits the two players need for transmitting this information is:

$$\log(\alpha_A(m) + 1) + \log(\alpha_B(m) + 1) < \log(2^{m-1}) + \log(2^m) = 2m - 1$$

In the full paper ([2]) we show that the new strategies forms an equilibrium.

Acknowledgments. The work of the first two authors was supported by a grant from the Israeli Academy of Sciences. The third author was supported by the National Science Foundation.

References

1. Liad Blumrosen and Noam Nisan. Auctions with severely bounded communications. In *43th FOCS*, 2002.
2. Liad Blumrosen, Noam Nisan, and Ilya Segal. Multi-player and multi-round auctions with severely bounded communications, 2003. Full version. available from <http://www.cs.huji.ac.il/~liad>.
3. Riley J. G. and Samuelson W. F. Optimal auctions. *American Economic Review*, pages 381–392, 1981.
4. Andrew V. Goldberg, Jason D. Hartline, and Andrew Wright. Competitive auctions and digital goods. In *Symposium on Discrete Algorithms*, pages 735–744, 2001.
5. Papadimitriou C. H. Algorithms, games, and the internet. *ACM Symposium on Theory of Computing*, 2001.
6. Paul Klemperer. The economic theory of auctions. *Edward Elgar Publishing*, 2000.
7. Ron Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *ACM Conference on Electronic Commerce*, pages 233–241, 2000.
8. Daniel Lehmann, Liadan Ita O’Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *1st ACM conference on electronic commerce*, 1999.
9. R. B. Myerson. Optimal auction design. *Mathematical of operational research*, pages 58–73, 1981.
10. Noam Nisan. Algorithms for selfish agents. In *STACS*, 1999.
11. Noam Nisan, Shmulik London, Ori Regev, and Noam Camiel. Globally distributed computation over the internet – the popcorn project. In *ICDCS*, 1998.
12. Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *STOC*, 1999.
13. Noam Nisan and Ilya Segal. Communication requirements of efficiency and supporting lindahl prices, 2003. working paper available from <http://www.cs.huji.ac.il/~noam/mkts.html>.
14. Web page. ebay. <http://www.ebay.com>.
15. Web page. vertical-net. <http://www.verticalnet.com>.
16. Amir Ronen and Amin Saberi. Optimal auctions are hard. In *43th FOCS*, 2002.
17. W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, pages 8–37, 1961.
18. Rakesh Vohra and Sven de Vries. Combinatorial auctions: A survey, 2000. Available from www.kellogg.nwu.edu/faculty/vohra/htm/res.htm.
19. C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2), 1992.
20. W.E. Walsh, M.P. Wellman, P.R. Wurman, and J.K. MacKie-Mason. Auction protocols for decentralized scheduling. In *Proceedings of The Eighteenth International Conference on Distributed Computing Systems (ICDCS-98)*, 1998.
21. Web-page. commerce-one. <http://www.commerceone.com>.

Network Lifetime and Power Assignment in ad hoc Wireless Networks

Gruia Calinescu¹, Sanjiv Kapoor², Alexander Olshevsky³, and
Alexander Zelikovsky⁴

¹ Department of Computer Science, Illinois Institute of Technology,
Chicago, IL 60616
`calinesc@iit.edu`

² Department of Computer Science, Illinois Institute of Technology,
Chicago, IL 60616.
`kapoor@iit.edu`

³ Department of Electrical Engineering, Georgia Institute of Technology,
Atlanta, GA 30332
`gte864k@prism.gatech.edu`

⁴ Computer Science Department, Georgia State University,
Atlanta, GA 30303
`alexz@cs.gsu.edu`

Abstract. Used for topology control in ad-hoc wireless networks, Power Assignment is a family of problems, each defined by a certain connectivity constraint (such as strong connectivity). The input consists of a *directed* complete weighted graph $G = (V, c)$. The *power* of a vertex u in a directed spanning subgraph H is given by $p_H(u) = \max_{uv \in E(H)} c(uv)$. The *power* of H is given by $p(H) = \sum_{u \in V} p_H(u)$. Power Assignment seeks to minimize $p(H)$ while H satisfies the given connectivity constraint. We present asymptotically optimal $O(\log n)$ -approximation algorithms for three Power Assignment problems: Min-Power Strong Connectivity, Min-Power Symmetric Connectivity (the undirected graph having an edge uv iff H has both uv and vu must be connected) and Min-Power Broadcast (the input also has $r \in V$, and H must be a r -rooted outgoing spanning arborescence).

For Min-Power Symmetric Connectivity in the Euclidean with efficiency case (when $c(u, v) = \|u, v\|^\kappa / e(u)$, where $\|u, v\|$ is the Euclidean distance, κ is a constant between 2 and 5, and $e(u)$ is the transmission efficiency of node u), we present a simple constant-factor approximation algorithm. For all three problems we give exact dynamic programming algorithms in the Euclidean with efficiency case when the nodes lie on a line.

In Network Lifetime, each node u has an initial battery supply $b(u)$, and the objective is to assign each directed subgraph H satisfying the connectivity constraint a real variable $\alpha(H) \geq 0$ with the objective of maximizing $\sum_H \alpha(H)$ subject to $\sum_H p_T(u) \alpha(H) \leq b(u)$ for each node $u \in V$. We are the first to study Network Lifetime and give approximation algorithms based on the PTAS for packing linear programs of Garg and Könemann. The approximation ratio for each case of Network Lifetime is equal to the approximation ratio of the corresponding Power Assignment problem with non-uniform transmission efficiency.

1 Introduction

Energy efficiency has recently become one of the most critical issues in routing of ad-hoc networks. Unlike wired networks or cellular networks, no wired backbone infrastructure is installed in ad hoc wireless networks. A communication session is achieved either through single-hop transmission or by relaying through intermediate nodes otherwise. In this paper we consider a static ad-hoc network model in which each node is supplied with a certain number of batteries and an omnidirectional antenna. For the purpose of energy conservation, each node can adjust its transmitting power, based on the distance to the receiving node and the background noise. Our routing protocol model assumes that each node periodically retransmit the hello-message to all its neighbors in the prescribed transmission range.

Formally, let $G = (V, E, c)$ be a weighted directed graph on network nodes with a *power requirement* function $c : E \rightarrow R^+$ defined on the edges. Given a *power assignment* function $p : V \rightarrow R^+$, a directed edge (u, v) is *supported* by p if $p(u) \geq c(u, v)$. The supported subgraph (sometimes called in the literature "transmission graph") H of G consists of supported edges. We consider the following network connectivity constraints (sometimes called in the literature "topology requirements") Q for the graph H : (1) strong connectivity, when H is strongly connected; (2) symmetric connectivity, when the undirected graph having an edge uv iff H has both uv and vu must be connected (3) broadcast (resp. multicast) from a root $r \in V$, when H contains a directed spanning tree rooted at r (resp. directed Steiner tree for given subset of nodes rooted at r). In this paper we start by considering the following generic optimization formulation [1,2].

Power Assignment problem. Given a power requirement graph $G = (V, E, c)$ and a connectivity constraint Q , find power assignment $p : V \rightarrow R^+$ of the minimum total power $\sum_{v \in V} p(v)$ such that the supported subgraph H satisfies the given connectivity constraint Q .

For simplicity of exposition, we use mostly the following equivalent definition of the Power Assignment problem: Given a directed spanning subgraph H , define the *power* of a vertex u as $p_H(u) = \max_{uv \in E(H)} c(uv)$ and the *power* of H as $p(H) = \sum_{u \in V} p_H(u)$. To see the equivalence, note that an optimal power assignment supporting directed spanning subgraph H never has $p(v) > \max_{uv \in E(H)} c(uv)$. Then the Power Assignment problem becomes finding the directed spanning subgraph H satisfying the connectivity constraint with minimum $p(H)$. Specifying the connectivity constraint, we obtain the following problems: Min-Power Strong Connectivity, Min-Power Symmetric Connectivity, Min-Power Broadcast, and Min-Power Multicast.

Although the Power Assignment problem formulation is quite relevant to the power-efficient routing it disregards possibly different number of batteries initially available to different nodes and, more importantly, the possibility of dynamic readjustment of the power assignment. In this paper we introduce a new power assignment formulation with a more relevant objective of maximizing the time period the network connectivity constraint is satisfied.

Formally, we assume that each node $v \in V$ is initially equipped with a battery supply $b(v)$ which is reduced by amount of $t \cdot p(v)$ for each time period t during which v is assigned power $p(v)$. A *power schedule* PT is a set of pairs (p_i, t_i) , $i = 1, \dots, m$, of power assignments $p_i : V \rightarrow R^+$ and time periods t_i during which the power assignment p_i is used. We say that the power schedule PT is *feasible* if the total amount of energy used by each node v during the entire schedule PT does not exceed its initial battery supply $b(v)$, i.e., $\sum_{i=1}^m t_i \cdot p_i(v) \leq b(v)$.

Network Lifetime problem. Given a power requirement graph $G = (V, E, c)$, a battery supply $b : V \rightarrow R^+$ and a connectivity constraint Q , find a feasible power schedule $PT = \{(p_1, t_1), \dots, (p_n, t_m)\}$ of the maximum total time $\sum_{i=1}^m t_i$ such that for each power assignment p_i , the supported subgraph H satisfies the given connectivity constraint Q .

Using the equivalent formulation, Network Life problem becomes the following linear programming problem: each directed subgraph H satisfying the connectivity constraint is assigned a real variable $\alpha(H) \geq 0$ with the objective of maximizing $\sum_H \alpha(H)$ subject to $\sum_H p_T(u) \alpha(H) \leq b(u)$ for each node $u \in V$. We note that an solution with only $|V|$ non-zero variables $\alpha(H)$ exists, show that Network Life is NP-hard under several connectivity constraints, and give the first approximation algorithms for Network Life based on the PTAS for packing linear programs of Garg and Könemann [3].

The related problem considered by Cardei et al [4] has uniform unadjustable power assignments with the objective to maximize number of disjoint dominating sets in a graph. The drawback of this formulation is that dominating sets are required to be disjoint while dropping this requirement will give better solution for the original problem. S. Slijepcevic and M. Potkonjak [5] and Cardei and Du [4] discuss the construction of disjoint set covers with the goal of extending the lifetime of wireless sensor networks. The sets are disks given by the sensor unadjustable range, and the elements to be covered are a fixed set of targets. A similar problem but in a different model has been studied by Zussman and Segall [6]. They assume that the most of energy consumption of wireless networks comes from routing the traffic, rather than routing control messages. They look for the best traffic flow routes for a given set of traffic demands using concurrent flow approaches [7] for the case when nodes do not have adjustable ranges.

Besides the general case of the given power requirements graph G , we consider the following important special cases : (1) symmetric case, where $c(u, v) = c(v, u)$; 2) Euclidean case, where $c(u, v) = d(u, v)^\kappa$, where $d(u, v)$ the Euclidean distance between u and v and κ is the signal attenuation exponent, which is assumed to be in between 2 and 5 and is the same for all pairs of nodes; (3) single line case, which is the subcase of Euclidean case when all nodes lie on a single line.

We also consider the following very important way of generating an asymmetric power requirement graph G' from a given symmetric power requirement graph G . Let $e : V \rightarrow R^+$ be the *transmission efficiency* defined on nodes of G , then power requirements with non-uniform transmission efficiency $G' = (V, E, c')$ are defined as $c'(u, v) = c(u, v)/e(u)$. This definition is motivated by possible

co-existence of heterogenous nodes and by our solution method for Network Lifetime. We also consider the three special cases above with non-uniform transmission efficiency, while the asymmetric power requirements case is not changed by the addition of non-uniform transmission efficiency.

Table 1. Table of upper bounds (UB) and lower bounds (LB) on the Power Assignment complexity. New results are bold. Marked by * are the folklore results, while references preceded by ** indicate the result is implicit in the respective papers.

power requirements	Complexity of the Power Assignment problem					
	asymmetric		Euclidean+eff.		symmetric	
Conn. Constraints	UB	LB	UB	LB	UB	LB
Strong Conn.	$3 + 2 \ln(n-1)$	SCH	$3 + 2 \ln(n-1)$	NPH	2 [8,9]	MAX-SNP*
Broadcast	$2 + 2 \ln(n-1)$	SCH	$2 + 2 \ln(n-1)$	NPH	$2 + 2 \ln(n-1)$	SCH [11,1]
Multicast	DST*	DSTH	DST*	NPH	$O(\ln n)$ ** [12]	SCH** [11,1]
Symmetric Conn.	$2 + 2 \ln(n-1)$	SCH	11.73	NPH	$\frac{5}{3} + \epsilon$ [13]	MAX-SNPH*

We present most of our new results on Power Assignment in Table 1, together with some of the existing results. For a more comprehensive survey of existing results, we refer to [15]. We omit the case of a single line – then all enlisted problems can be solved exactly in polynomial time. More precise, without efficiency, the algorithms were folklore or appeared in [9], and with efficiency we claim polynomial time algorithms.

SCH is used to mean as hard as Set Cover; based on the Feige [16] result there is no polynomial-time algorithm with approximation ratio $(1 - \epsilon) \ln n$ for any $\epsilon > 0$ unless $P = NP$. DST means that the problem reduces (approximation-preserving) to Directed Steiner Tree and DSTH means Directed Steiner Tree reduces (approximation-preserving) to the problem given by the cell. Best known approximation ratio for Directed Steiner Tree is $O(n^\epsilon)$ for any $\epsilon > 0$ and finding a poly-logarithmic approximation ratio remains a major open problem in approximation algorithms.

Liang [22] considered some asymmetric power requirements and presented, among other results, the straightforward approximation-preserving reduction (which we consider folklore, and is implicit, for example, in [12]) of Min-Power Broadcast and Min-Power Multicast to Directed Steiner Tree. We improve the approximation ratio for Min-Power Broadcast to $2 + 2 \ln(n-1)$. Min-Power Symmetric Connectivity and Min-Power Strong Connectivity were not considered before with asymmetric power requirements. For Min-Power Broadcast with symmetric power requirements we improve the approximation ratio from $10.8 \ln n$ of [12] to $2 + 2 \ln(n-1)$. We remark that the method of [12] also works for Multicast with symmetric power requirements, giving a $O(\ln n)$ approximation ratio, while with asymmetric power requirements, the problem appears to be harder - it is DSTH to be precise.

The rest of the paper is organized as follows. In Section 2 we use methods designed for Node Weighted Steiner Trees to give $O(\ln n)$ approximation algorithms for Min-Power Broadcast, and Min-Power Strong Connectivity, all with

asymmetric power requirements (Min-Power Symmetric Connectivity is omitted due to space limitations). In Section 3 we give constant-factor approximations for symmetric connectivity in the Euclidean with efficiency case. Section 4 deals with the Network Lifetime problem. Section 5 lists extensions of this work and some remaining open problems for Power Assignment. Due to space limitations we omit our results on lower bounds on the approximation complexity of the Power Assignment problem and dynamic programming algorithms for the case of a single line with efficiency.

2 Algorithms for Asymmetric Power Requirements

In this section we assume the power requirements are asymmetric and arbitrary. We present the algorithm for Min-Power Broadcast with an asymptotically optimal $2(1 + \ln(n - 1))$ approximation ratio, where n is the cardinality of the vertex set. The algorithm is greedy and we adopt the technique used for Node Weighted Steiner Trees by [17], which in turn is using an analysis of the greedy set cover algorithm different than the standard one of Chvatal [18]. The algorithm attempts to reduce the "size" of the problem by greedily adding structures.

The algorithm starts iteration i with a directed graph H_i , seen as a set of arcs with vertex set V . The strongly connected components of H_i which do not contain the root and have no incoming arc are called *unhit components*. The algorithm stops if no unhit components exists, since in this case the root can reach every vertex in H_i . Otherwise, a weighted structure which we call *spider* (details below) is computed such that it achieves the biggest reduction in the number of unhit components divided by the weight of the spider. The algorithm then adds the spider (seen as a set of arcs) to H_i to obtain H_{i+1} . For an arc $uv \in E(G)$, we use *cost* to mean $c(uv)$, the power requirement of the arc.

Definition 1. *A spider is a directed graph consisting of one vertex called head and a set of directed paths (called legs), each of them from the head to a (vertices called) feet of the spider. The definition allows legs to share vertices and arcs. The weight of the spider S , denoted by $w(S)$, is the maximum cost of the arcs leaving the head plus the sum of costs of the legs, where the cost of a leg is the sum of the costs of its arcs without the arc leaving the head.*

See Figure 1 for an illustration of a spider and its weight. The weight of the spider S can be higher than $p(S)$ (here we assume S is a set of arcs), as the legs of the spider can share vertices, and for those vertices the sum (as opposed to the maximum) of the costs of outgoing arcs contributes to $w(S)$. From every unhit component of H_i we arbitrarily pick a vertex and we call it a *representative*.

Definition 2. *The shrink factor $sf(S)$ of a spider S with head h is either the number of representatives among its feet if h is reachable (where, by convention, a vertex is reachable from itself) from the root or if h is not reachable from any of its feet, or the number of representatives among its feet minus one, otherwise.*

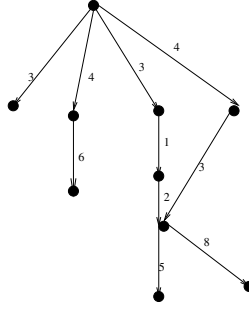


Fig. 1. A spider with four legs, weight $\max\{3, 4, 3, 4\} + 6 + (1 + 2 + 5) + (3 + 8) = 29$ and power 25.

Input: A complete directed graph $G = (V, E)$ with power requirement function $c(u, v)$ and a root vertex

Output: An directed spanning graph H (seen as a set of arcs, with $V(H) = V$) such that in H there is a path from the root to every vertex of V .

- (1) Initialize $H = \emptyset$
 - (2) While H has at least one unhit component
 - (2.1) Find the spider S which minimizes $w(S)/(sf(S))$ with respect to H
 - (2.2) Set $H \leftarrow H \cup S$
-

Fig. 2. The Greedy Algorithm for Min-Power Broadcast with asymmetric power requirements

Our algorithm appears in Figure 2. We describe later the detailed implementation of Step 2.1 of the algorithm. Let $u(H)$ be the number of unhit components of direct graph H . Due to space limitations, we omit the proof of the next lemma:

Lemma 1. *For a spider S (seen as a set of arcs), $u(H_i \cup S) \leq u(H_i) - sf(S)$.*

Fact 1 *Given a spider S (seen as a set of arcs), $p(H_i \cup S) \leq p(H_i) + w(S)$.*

Next we describe how to find the spider which minimizes its weight divided by its shrink factor. In fact, we search for *powered* spiders, which besides head h and legs have a fixed power $p(h)$ associated with the head. The weight of the powered spider S' , denoted by $w(S')$, equals $p(h)$ plus the sum of costs of the legs (where as before the cost of a leg is the sum of the costs of its arcs without the arc leaving the head). Given a spider one immediately obtains a powered spider of the same weight, while given a powered spider S' , the spider S obtained from S' by keeping only the edges of S' (thus ignoring the fixed power of the head) satisfies $w(S) \leq w(S')$.

We try all possible heads h , and all possible discrete power for the head (there are at most n such discrete power values - precisely the values $c(hu)$ for every $u \in G$, where $c(hh) = 0$ by convention). Define the *children* of the head to be the vertices within its power value - where the head is also considered a child. For each representative r_i , compute the shortest path P_i from a child of h to r_i . If h is not reachable from the root, partition the representatives in two sets - R_1 which cannot reach h and R_2 which can reach h ; otherwise let $R_1 = R$ and $R_2 = \emptyset$. Sort R_1 and R_2 such that the lengths of the paths P_i are in nondecreasing order. Then the best spider with head h and the given power value can be obtained by trying all $0 \leq j_1 \leq |R_1|$ and $0 \leq j_2 \leq |R_2|$ and taking the paths P_i leading to the first j_1 representatives of R_1 and the first j_2 representatives of R_2 .

The following lemma shows the existence of a good spider; it is a counterpart of Lemma 4.1 and Theorem 3.1 of [17]. Let OPT denote the value of the optimum solution.

Lemma 2. *Given any graph H_i and set of representatives obtained from H_i , there is a spider S such that $\frac{w(S)}{sf(S)} \leq 2 \frac{OPT}{u(H_i)}$.*

Proof. Let T be the optimum arborescence outgoing from the root and R the set of representatives obtained from H_i ; $|R| = u(H_i)$. Traverse T in postorder and whenever a vertex v is the ancestor of at least two representatives (where by default every vertex is an ancestor of itself) define a spider with head v and legs given by the paths of T from v to the representatives having v as an ancestor. Remove v and its descendents from T , and repeat. The process stops if the number of remaining representatives is less than two. If there is one representative left, define one last spider with the head the root and one leg to the remaining representative. Let S_i , for $1 \leq i \leq q$ be the spiders so obtained.

It is immediate that $w(S_1) + w(S_2) + \dots + w(S_q) \leq OPT$. If $r(S_i)$ is the number of representatives in spider S_i , we have that $r(S_1) + r(S_2) + \dots + r(S_q) = |R|$. Note that $r(S_i) \leq 2sf(S_i)$, as except for the spider with the root as its head (for which $r(S_i) = sf(S_i)$) $2 \leq r(S_i) \leq sf(S_i) + 1$. We conclude that $2(sf(S_1) + sf(S_2) + \dots + sf(S_q)) \geq |R| = u(H_i)$. The spider with highest ratio among S_j , $1 \leq j \leq q$, has $\frac{w(S_j)}{2sf(S_j)} \leq \frac{OPT}{u(H_i)}$.

Theorem 1. *The algorithm described in this subsection has approximation ratio $2(1 + \ln(n - 1))$ for Min-Power Broadcast with asymmetric power requirements.*

Proof. Let q_i be the number of unhit components of H_i (where H_0 is the initial graph with no edges), S_i be the spider picked to be added to H_i , $d_i = sf(S_i)$, and $w_i = w(S_i)$.

From Lemma 1, we have: $q_{i+1} \leq q_i - d_i$. Since the algorithm is greedy, by Lemma 2, $\frac{w_i}{d_i} \leq \frac{2OPT}{q_i}$. Plugging equation the above equations into each other and rearranging the terms, it follows that $q_{i+1} \leq q_i - d_i \leq q_i(1 - \frac{w_i}{2OPT})$. Assuming there are m steps, this implies that $q_{m-1} \leq q_0 \prod_{k=0}^{m-2} (1 - \frac{w_k}{2OPT})$. Taking natural logarithm on both sides and using the inequality $\ln(1 + x) \leq x$,

we obtain that $\ln \frac{q_0}{q_{m-1}} \geq \frac{\sum_{k=0}^{m-2} w_k}{2OPT}$. However, $q_{m-1} \geq 1$ and $q_0 = n - 1$ so that $2OPT \ln(n - 1) \geq \sum_{k=0}^{m-2} w_k$.

The weight of the last spider can be bounded as $w_{m-1} \leq 2OPT$ from Lemma 2. Finally, since $APPROX \leq \sum_{k=0}^{m-1} w_k$, which follows from Fact 1, we have that $APPROX \leq 2(1 + \ln(n - 1))OPT$.

2.1 Min-Power Strong Connectivity with Asymmetric Power Requirements

In this subsection we use the previous result to give an approximation algorithm for Min-Power Strong Connectivity with asymmetric power requirements. Let v be an arbitrary vertex. An optimum solution of power OPT contains an outgoing arborescence A_{out} rooted at v (so $p(A_{out}) \leq OPT$) and an incoming arborescence A_{in} rooted at v (so $c(A_{in}) = p(A_{in}) \leq OPT$).

The broadcast algorithm in the previous subsection produces an outgoing arborescence B_{out} rooted at v with $p(B_{out}) \leq 2(1 + \ln(n - 1))p(A_{out})$. Edmonds' algorithm produces a minimum cost arborescence B_{in} rooted at v with $c(B_{in}) \leq c(A_{in})$. Then $p(B_{out} \cup B_{in}) \leq p(B_{out}) + c(B_{in}) \leq 2(1 + \ln(n - 1))p(A_{out}) + c(A_{in}) \leq (2 \ln(n - 1) + 3)OPT$. Therefore we have

Theorem 2. *There is a $2 \ln(n - 1) + 3$ -approximation algorithm for Strong Connectivity with asymmetric power requirements.*

We mention that Min-Power Unicast with asymmetric power requirements is solved by a shortest paths computation. Min-Power Symmetric Unicast (where the goal is to obtain the minimum power undirected path connecting two given vertices) with asymmetric power requirements can also be solved in $O(n^2 \log n)$ by a shortest paths computation in a specially constructed graph described in Section 4 of [2]. Algorithms faster than $O(n^2)$ are not known for Min-Power Symmetric Unicast even in the simplest Line case.

3 Min-Power Symmetric Connectivity in the Euclidean-with-Efficiency Case

In this section we present a constant-ratio algorithm for Min-Power Symmetric Connectivity when power requirements are in the Euclidean-with-efficiency model: $c(uv) = d(u, v)^\kappa / e(u)$, where d is the Euclidean distance and $2 \leq \kappa \leq 5$.

The algorithm is very simple: for any unordered pair of nodes uv define $w(u, v) = c(u, v) + c(v, u)$ and compute as output a minimum spanning tree M in the resulting weighted undirected graph.

We prove the algorithm above (which we call the *MST* algorithm) has constant approximation ratio using only the fact that d is an arbitrary metric (as for example in the three dimensional Euclidean case).

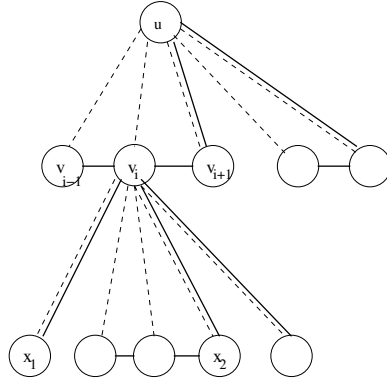


Fig. 3. An illustration of the transformation from T (the dotted lines) to T_r (given by solid lines).

For any tree T , let $w(T) = \sum_{(u,v) \in T} w(u, v)$. Note that

$$w(T) = \sum_{v \in V} \sum_{y: (v,y) \in T} c(v, y) \geq \sum_{v \in V} \max_{y: (v,y) \in T} c(v, y) = p(T).$$

Let T be an arbitrary spanning tree of G . We arbitrarily pick a root for T . For each node u with $k(u)$ children, we sort the children $v_1, v_2, \dots, v_{k(u)}$ such that $d(u, v_i) \geq d(u, v_{i+1})$. With a fixed parameter $r > 1$ (to be chosen later), we modify T in a bottom-up manner by replacing, for each $1 \leq i < k(u)$, each edge (u, v_i) with (v_i, v_{i+1}) if $d(u, v_i) \leq r \cdot d(u, v_{i+1})$ (see Figure 3). We denote by T_r the rooted resulting tree. Our main lemma (whose proof we omit due to space constraint) below relates the weight of T_r to the power of T :

Lemma 3. *For any rooted tree T , $w(T_r) \leq \left(2^\kappa + (r+1)^\kappa + \frac{r^\kappa}{r^\kappa - 1}\right) p(T)$*

Note that $p(MST) \leq w(MST) \leq w(T_r) \leq \left(2^\kappa + (r+1)^\kappa + \frac{r^\kappa}{r^\kappa - 1}\right) p(T)$, where T is the minimum power tree.

Theorem 3. *The approximation ratio of the MST algorithm is at most $\min_{r>1} \{2^\kappa + (r+1)^\kappa + \frac{r^\kappa}{r^\kappa - 1}\}$*

Numerically obtained, this approximation ratio is (i) 11.73 for $\kappa = 2$, achieved at $r = 1.32$ (ii) 20.99 for $\kappa = 3$, achieved at $r = 1.15$; (iii) 38.49 for $\kappa = 4$, achieved at $r = 1.08$ (iv) 72.72 for $\kappa = 5$, achieved at $r = 1.05$.

4 Network Lifetime

In this section we first show that the Network Lifetime problem is NP-Hard for symmetric power requirements and each considered connectivity constraint:

strong connectivity, symmetric connectivity and broadcast. Then we show how the Garg-Köneman PTAS [3] PTAS can be used for reducing Network Lifetime to Power Assignment. In the following we drop mentioning the specific connectivity constraint when the discussion applies to all possible connectivity constraints.

Recall that the Network Lifetime problem has as input a power requirement graph $G = (V, E, c)$ and a battery supply vector $b : V \rightarrow R^+$. A set $\mathcal{S}$ of directed spanning subgraphs of G is given implicitly by the connectivity constraints. In general, $|\mathcal{S}|$ is exponential in $|V|$. Then Network Lifetime is the following packing linear program: Maximize $\sum_{H \in \mathcal{S}} x_H$ subject to $\sum_{H \in \mathcal{S}} p_H(v) x_H \leq b(v), \forall v \in V, x_H \geq 0, \forall H \in \mathcal{S}$.

We note that an optimum vertex solution only uses $|V|$ non-zero variables x_H . With potentially exponential number of columns, it is not surprising the following theorem, whose proof uses an idea from [4] and is omitted due to space limitations, holds:

Theorem 4. *Even in the special case when all the nodes have the same battery supply, the Network Lifetime for Symmetric Connectivity (or Broadcast or Strong Connectivity) problem is NP-hard in the symmetric power requirements case.*

The Network Lifetime linear program above is a packing LP. In general, a packing LP is defined as

$$\max\{c^T x \mid Ax \leq b, x \geq 0\} \quad (1)$$

where A, b , and c have positive entries; we denote the dimensions of A as $n \times l$. In our case the number of columns of A is prohibitively large (exponential in number of nodes) and we will use the $(1 + \epsilon)$ -approximation Garg-Köneman algorithm [3]. The algorithm assumes that the LP is implicitly given by a vector $b \in R^n$ and an algorithm which finds the column of A minimizing so-called length. The *length* of column j with respect to LP in Equation (1) and non-negative vector y is defined as $length_y(j) = \frac{\sum_{i=1}^n A(i,j)y(i)}{c(j)}$.

We cannot directly apply the Garg-Köneman algorithm because, as we notice below, the problem of finding the minimum length column is NP-Hard in our case, and we can only approximate the minimum length column. Fortunately, it is not difficult to see that when the Garg-Köneman $(1 + \epsilon)$ -approximation algorithm uses f -approximation minimum length columns it gives an $(1 + \epsilon)f$ -approximation solution to the packing LP (1) [19]¹.

The Garg-Köneman algorithm with f -approximate columns is presented in Figure 4. When applied to the Network Lifetime LP, it is easy to see that the problem of finding the minimum length column, corresponds to finding the minimum power assignment with transmission efficiencies inverse proportional to the elements of vector y , i.e., for each node $i = 1, \dots, n$, $e(i) = 1/y_i$. This implies the following general result.

¹ Although this complexity aspect has not been published anywhere in literature, it involves only a trivial modification of [3] and will appear in its journal version [19].

Input: A vector $b \in R^n$, $\epsilon > 0$, and an f -approximation algorithm F for the problem of finding the minimum length column $A_{j(y)}$ of a packing LP $\{\max c^T x | Ax \leq b, x \geq 0\}$

Output: A set of S of columns of A : $\{A_j\}_{j \in S}$ each supplied with the value of the corresponding variable x_j , such that x_j , for $j \in S$, are all non-zero variables in a feasible approximate solution of the packing LP $\{\max c^T x | Ax \leq b, x \geq 0\}$

- (1) Initialize: $\delta = (1 + \epsilon)((1 + \epsilon)n)^{-1/\epsilon}$, for $i = 1, \dots, n$ $y(i) \leftarrow \frac{\delta}{b(i)}$, $D \leftarrow n\delta$, $S \leftarrow \emptyset$.
 - (2) While $D < 1$
 - Find the column A_j ($j = j(y)$) using the f -approximate algorithm F .
 - Compute p , the index of the row with the minimum $\frac{b(i)}{A_j(i)}$
 - if $j \notin S$ $x_j \leftarrow \frac{b(p)}{A_q(p)}$ else $x_j \leftarrow x_j + \frac{b(p)}{A_q(p)}$
 - $S \leftarrow S \cup \{j\}$
 - For $i = 1, \dots, n$, $y(i) \leftarrow y(i) \left(1 + \epsilon \frac{b(p)}{A_j(p)} / \frac{b(i)}{A_j(i)}\right)$, $D \leftarrow b^T y$.
 - (3) Output $\{(j, \frac{x_j}{\log_{1+\epsilon} \frac{1}{\delta}})\}_{j \in S}$
-

Fig. 4. The Garg-Köneman Algorithm with f -approximate minimum length columns

Theorem 5. *For a connectivity constraint and a case of the power requirements graph, given an f -approximation algorithm F for Power Assignment with the given connectivity constraint and the case of the power requirements graph with added non-uniform efficiency, there is a $(1 + \epsilon)f$ -approximation algorithm for the corresponding Network Lifetime problem.*

The above theorem implies approximation algorithms for the Network Lifetime problem in the cases for which we developed approximation algorithms for the Power Assignment problem with nonuniform efficiency (see Table 1).

5 Conclusions

We believe the following results hold, but their exposition will complicate this long paper too much:

1. Min-Power Steiner Symmetric Connectivity with asymmetric power requirements, in which a given set of terminals must be symmetrically connected, can also be approximated with a $O(\log n)$ ratio using a spider structure similar to the one used for broadcast, but with a "symmetric" weight, and a greedy algorithm.
2. The algorithms for Node Weighted Steiner Tree of Guha and Khuller [20] can also be adapted (but in a more complicated way, as they are more complicated than [17]) to obtain, for any $\epsilon > 0$, algorithms with approximation ratio of $(1.35 + \epsilon) \ln n$ for Min-Power Symmetric Connectivity, Min-Power Steiner Symmetric Connectivity, Min-Power Broadcast, and Min-Power Strong Connectivity with asymmetric power requirements.

We leave open the existence of efficient exact or constant factors algorithm for Min-Power Broadcast or Min-Power Strong Connectivity in the Euclidean with efficiency case. We also leave open the NP-Hardness of Network Life in Euclidean cases.

Another special case is when nodes have non-uniform "sensitivity" $s(v)$. Even in the Line-with-sensitivity case, when $c(u, v) = \|u, v\|^\kappa / s(v)$, we do not know algorithms better than the general $O(\log n)$ algorithms from Section 2. Adding non-uniform sensitivity to symmetric power requirements results in Power Assignment problems as hard as set cover.

References

1. P. Wan, G. Calinescu, X.-Y. Li, and O. Frieder, "Minimum energy broadcast in static ad hoc wireless networks," *Wireless Networks*, 2002.
2. G. Calinescu, I. Mandoiu, and A. Zelikovsky, "Symmetric connectivity with minimum power consumption in radio networks," in *Proc. 2nd IFIP International Conference on Theoretical Computer Science, (TCS 2002)*, R. Baeza-Yates and U. Montaniri and N. Santoro (eds.), Kluwer Academic Publ., August 2002, 119–130.
3. N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *Proceedings of FOCS*, 1997.
4. D.-Z. D. M. Cardei, "Improving wireless sensor network lifetime through power aware organization," *Submitted to ACM Wireless Networks*.
5. S. Slijepcevic and M. Potkonjak, Power Efficient Organization of Wireless Sensor Networks, IEEE International Conference on Communications (ICC), Helsinki, June 2001, pp. 472–476.
6. G. Zussman and A. Segall, "Energy efficient routing in ad hoc disaster recovery networks," in *IEEE INFOCOM'03*, 2003.
7. F. Leighton and S. Rao, "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms," *Journal of the ACM*, vol. 6, pp. 787–832, 1999.
8. W. Chen and N. Huang, "The strongly connecting problem on multihop packet radio networks," *IEEE Transactions on COmmunications*, vol. 37, pp. 293–295, 1989.
9. L. M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc, "Power consumption in packet radio networks," *Theoretical Computer Science*, vol. 243, pp. 289–305, 2000, preliminary version in STACS'97.
10. A. E. Clementi, P. Penna, and R. Silvestri, "On the power assignment problem in radio networks," *Electronic Colloquium on Computational Complexity*, vol. Report TR00-054, 2000, preliminary results in APPROX'99 and STACS'2000.
11. A. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca, "On the complexity of computing minimum energy consumption broadcast subgraphs," in *18th Annual Symposium on Theoretical Aspects of Computer Science, LNCS 2010*, 2001, pp. 121–131.
12. I. Caragiannis, C. Kaklamani and P. Kanellopoulos, "New results for energy-efficient broadcasting in wireless networks," in *ISAAC'2002*, 2002, pp. 332–343.
13. E. Althaus, G. Calinescu, I. Mandoiu, S. Prasad, N. Tchervenski and A. Zelikovsky, "Power efficient range assignment in ad-hoc wireless networks," *WCNC'03*, 2003, pp. 1889–1894.

14. D. Blough, M. Leoncini, G. Resta, and P. Santi, "On the symmetric range assignment problem in wireless ad hoc networks," in *2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*. Kluwer Academic Publishers, 2002, pp. 71–82.
15. A. Clementi, G. Huiban, P. Penna, G. Rossi, and Y. Verhoeven, "Some recent theoretical advances and open questions on energy consumption in ad-hoc wireless networks," in *Proc. 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE)*, 2002.
16. U. Feige, "A threshold of $\ln n$ for approximating set cover," *Journal of the ACM*, vol. 45, pp. 634–652, 1998.
17. P. Klein and R. Ravi, "A nearly best-possible approximation algorithm for node-weighted steiner trees," *Journal of Algorithms*, vol. 19, pp. 104–115, 1995.
18. V. Chvatal, "A greedy heuristic for the set covering problem," *Mathematics of Operation Research*, vol. 4, pp. 233–235, 1979.
19. J. Könneman, "Personal communication."
20. S. Guha and S. Khuller, "Improved methods for approximating node weighted steiner trees and connected dominating sets," *Information and Computation*, vol. 150, pp. 57–74, 1999.
21. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.
22. W. Liang, "Constructing Minimum-Energy Broadcast Trees in Wireless Ad Hoc Networks" *MOBIHOC'02*, 112–122, 2002.

Disjoint Unit Spheres admit at Most Two Line Transversals

Otfried Cheong¹, Xavier Goaoc², and Hyeon-Suk Na³

¹ Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. ocheong@win.tue.nl

² LORIA (INRIA Lorraine), 615, rue du Jardin Botanique, B.P. 101, 54602 Villers-les-Nancy, France. xavier.goaoc@loria.fr

³ School of Computing, Soongsil University, Seoul, South Korea. hsnaa@computing.ssu.ac.kr

Abstract. We show that a set of n disjoint unit spheres in $\mathbb{R}^d$ admits at most *two* distinct geometric permutations, or line transversals, if n is large enough. This bound is optimal.

1 Introduction

A line ℓ is a *line transversal* for a set $\mathcal{S}$ of pairwise disjoint convex bodies in $\mathbb{R}^d$ if it intersects every element of $\mathcal{S}$. A line transversal defines two linear orders on $\mathcal{S}$, namely the order in which ℓ intersects the bodies, where we can choose to orient ℓ in two directions. Since the two orders are essentially the same (one is the reverse of the other), we consider them as a single *geometric permutation*.

Bounds on the maximum number of geometric permutations were established about a decade ago: a tight bound of $2n - 2$ is known for $d = 2$ [2], for higher dimension the number is in $\Omega(n^{d-1})$ [6] and in $O(n^{2d-2})$ [10]. The gap was closed for the special case of spheres by Smorodinsky et al. [9], who showed that n spheres in $\mathbb{R}^d$ admit $\Theta(n^{d-1})$ geometric permutations. This result can be generalized to “fat” convex objects [8].

The even more specialized case of congruent spheres was treated by Smorodinsky et al. [9] and independently by Asinowski [1]. They proved that n unit circles in $\mathbb{R}^2$ admit at most two geometric permutations if n is large enough (the proof by Asinowski holds for all $n \geq 4$). Zhou and Suri established an upper bound of 16 for all d and n sufficiently large, a result quickly improved by Katchalski, Suri, and Zhou [7] and independently by Huang, Xu, and Chen [5] to 4. When the spheres are not congruent, but the ratio of the radii of the largest and smallest sphere is bounded by γ , then the number of geometric permutations is bounded by $O(\gamma^{\log \gamma})$ [12].

Katchalski et al. show that for n large enough, two line transversals can make an angle of at most $O(1/n)$ with each other, so all line transversals are “essentially” parallel. They define a *switched pair* to be a pair of spheres (A, B) such that there are two line transversals ℓ and ℓ' (for all n spheres) where ℓ visits A before B , while ℓ' visits B before A . Katchalski et al. prove that any sphere

can participate in at most one switched pair, and that the two spheres forming a switched pair must appear consecutively in any geometric permutation of the set. It follows that any two geometric permutations differ only in that the elements of some switched pair may have been exchanged. Katchalski et al.'s main result is that there are at most two switched pairs in a set of n disjoint unit spheres, implying the bound of four geometric permutations.

We show that in fact there cannot be more than one switched pair. This implies that, for n large enough, a set of n disjoint unit spheres admits at most two geometric permutations, which differ only by the swapping of two adjacent elements. Since there are arbitrarily large sets of unit spheres in $\mathbb{R}^d$ with one switched pair, this bound is optimal.

Surveys of geometric transversal theory are Goodman et al. [3] and Wenger [11]. The latter also discusses Helly-type theorems for line transversals. A recent result in that area by Holmsen et al. [4] proves the existence of a number $n_0 \leq 46$ such that the following holds: Let $\mathcal{S}$ be a set of disjoint unit spheres in $\mathbb{R}^3$. If every n_0 members of $\mathcal{S}$ have a line transversal, then $\mathcal{S}$ has a line transversal. Our present results slightly simplify the proof of this result.

2 The Proof

A *unit sphere* is a sphere of radius 1. We say that two unit spheres are *disjoint* if their interiors are (in other words, we allow the spheres to touch). A line *stabs* a sphere if it intersects the closed sphere (and so a tangent to a sphere stabs it). A *line transversal* for a set of disjoint unit spheres is a line that stabs all the spheres, with the restriction that it is not allowed to be tangent to two spheres in a common point (as such a line does not define a geometric permutation).

Given two disjoint unit spheres A and B , let $g(A, B)$ be their center of gravity and $\Pi(A, B)$ be their bisecting hyperplane. If the centers of A and B are a and b , then $g(A, B)$ is the mid-point of a and b , and $\Pi(A, B)$ is the hyperplane through $g(A, B)$ orthogonal to the line ab .

We first repeat a basic lemma by Katchalski et al.

Lemma 1. [7, Lemma 2.3] *Let ℓ and ℓ' be two different line transversals of a set $\mathcal{S}$ of n disjoint unit spheres in $\mathbb{R}^d$. Then the angle between the direction vectors of ℓ and ℓ' is $O(1/n)$.*

Proof. A volume argument shows that the distance between the first and last sphere stabbed by ℓ is $\Omega(n)$. Since ℓ and ℓ' have distance at most 2 over an interval of length $\Omega(n)$, their direction vectors make an angle of $O(1/n)$.

Lemma 1 implies that all line transversals for a set of spheres are nearly parallel.

We continue with a warm-up lemma in two dimensions.

Lemma 2. *Let S and T be two unit-radius disks in $\mathbb{R}^2$ with centers $(-\lambda, 0)$ and $(\lambda, 0)$, where $\lambda \geq \cos \beta$ for some angle β with $0 < \beta \leq \pi/2$. Then $S \cap T$ is contained in the ellipse*

$$\left(\frac{x}{\sin^2 \beta}\right)^2 + \left(\frac{y}{\sin \beta}\right)^2 \leq 1.$$

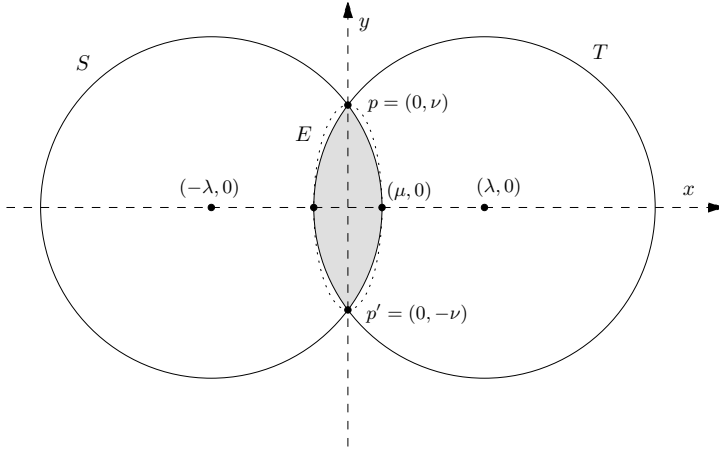


Fig. 1. The intersection of two disks is contained in an ellipse.

Proof. Let $(\mu, 0)$ and $(0, \nu)$ be the rightmost and topmost point of $S \cap T$ (see Figure 1). Consider the ellipse E defined as

$$\left(\frac{x}{\mu}\right)^2 + \left(\frac{y}{\nu}\right)^2 \leq 1.$$

E intersects the boundary of S in $p = (0, \nu)$ and $p' = (0, -\nu)$, and is tangent to it in $(\mu, 0)$. An ellipse can intersect a circle in at most four points and the tangency counts as two intersections, and so the intersections at p and p' are proper and there is no further intersection between the two curves. This implies that the boundary of E is divided into two pieces by p and p' , with one piece inside S and one outside S . Since $(-\mu, 0)$ lies inside S , the right hand side of E lies outside S . Symmetrically, the left hand side of E lies outside T , and so $S \cap T$ is contained in E . It remains to observe that

$$\nu^2 = 1 - \lambda^2 \leq 1 - \cos^2 \beta = \sin^2 \beta,$$

so $\nu \leq \sin \beta$, and

$$\mu = 1 - \lambda \leq 1 - \cos \beta \leq 1 - \cos^2 \beta = \sin^2 \beta,$$

which proves the lemma.

We now show that a transversal for two spheres cannot pass too far from their common center of gravity. Here and in the following, $d(\cdot, \cdot)$ denotes the Euclidean distance of two points.

Lemma 3. *Given two disjoint unit spheres A and B in $\mathbb{R}^d$ and a line ℓ stabbing both spheres, let p be the point of intersection of ℓ and $\Pi(A, B)$, and let β be the angle between ℓ and $\Pi(A, B)$. Then*

$$d(p, g(A, B)) \leq \sin \beta.$$

Proof. Let a and b be the centers of A and B and let v be the direction vector of ℓ , that is, ℓ can be written as $\{p + \lambda v \mid \lambda \in \mathbb{R}\}$. We first argue that proving the lemma for $d = 3$ is sufficient. Indeed, assume $d > 3$ and consider the 3-dimensional subspace Γ containing ℓ , a , and b . Since we have $d(a, \ell) \leq 1$ and $d(b, \ell) \leq 1$, the line ℓ stabs the 3-dimensional unit spheres $A \cap \Gamma$ and $B \cap \Gamma$. And since $\pi/2 - \beta$ is the angle between two vectors in Γ , namely v and $b - a$, β is also the angle between ℓ and the two-dimensional plane $\Pi(A, B) \cap \Gamma$. So if the lemma holds in Γ , then it also holds in $\mathbb{R}^d$.

In the rest of the proof we can therefore assume that $d = 3$. We choose a coordinate system where $a = (0, 0, -\rho)$, $b = (0, 0, \rho)$ with $\rho \geq 1$, and $v = (\cos \beta, 0, \sin \beta)$. Then $\Pi := \Pi(A, B)$ is the xy -plane and $g := g(A, B) = (0, 0, 0)$. Consider the cylinders $\text{cyl}(A) := \{u + \lambda v \mid u \in A, \lambda \in \mathbb{R}\}$ and $\text{cyl}(B)$ defined accordingly. Since ℓ stabs A and B , we have $p \in \text{cyl}(A) \cap \text{cyl}(B) \cap \Pi$.

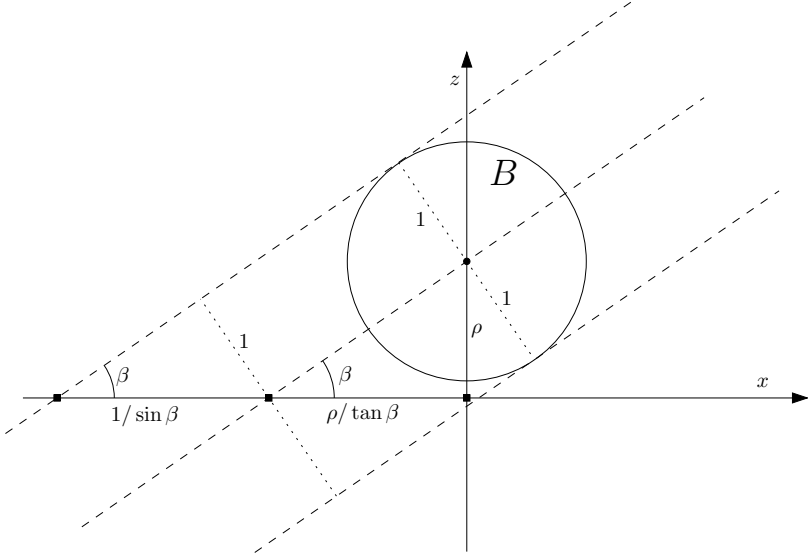


Fig. 2. The intersection of the cylinder with the xy -plane is an ellipse.

The intersection $B' := \text{cyl}(B) \cap \Pi$ is the ellipse (see Figure 2)

$$\sin^2 \beta \left(x + \frac{\rho}{\tan \beta}\right)^2 + y^2 \leq 1,$$

and symmetrically $A' := \text{cyl}(A) \cap \Pi$ is

$$\sin^2 \beta \left(x - \frac{\rho}{\tan \beta}\right)^2 + y^2 \leq 1.$$

If we let τ be the linear transformation

$$\tau : (x, y) \mapsto (x \sin \beta, y),$$

then $\tau(A')$ and $\tau(B')$ are unit-radius disks with centers $(\rho \cos \beta, 0)$ and $(-\rho \cos \beta, 0)$. By Lemma 2, the intersection $\tau(A' \cap B')$ is contained in the ellipse

$$\left(\frac{x}{\sin^2 \beta}\right)^2 + \left(\frac{y}{\sin \beta}\right)^2 \leq 1.$$

Applying τ^{-1} we find that $A' \cap B'$ is contained in the circle with radius $\sin \beta$ around g . Since $p \in A' \cap B'$, the lemma follows.

We now prove our key lemma.

Lemma 4. *Let A, B, C, D be four spheres from a set $\mathcal{S}$ of n disjoint unit spheres in $\mathbb{R}^d$, for n large enough. Assume there are two line transversals ℓ and ℓ' for $\mathcal{S}$, such that ℓ stabs the four spheres in the order $ABCD$, and ℓ' stabs them in the order $BADC$. Then*

$$d(g(A, B), g(C, D)) < 1 + O(1/n).$$

Proof. Let $\Pi_1 := \Pi(A, B)$, $\Pi_2 = \Pi(C, D)$, $g_1 := g(A, B)$, and $g_2 := g(C, D)$. We choose a coordinate system where Π_1 is the hyperplane $x_1 = 0$, and the intersection $\Pi_1 \cap \Pi_2$ is the subspace $x_1 = x_2 = 0$. We can make this choice such that the x_1 -coordinate of the center of A is < 0 , and that the x_2 -coordinate of the center of C is less than the x_2 -coordinate of the center of D . We can also assume that the x_2 -coordinate of g_1 is ≥ 0 (otherwise we swap A with B , C with D , and ℓ with ℓ'). Figure 3 shows the projection of the situation on the $x_1 x_2$ -plane.

Let $p_i := \ell \cap \Pi_i$, $p'_i := \ell' \cap \Pi_i$, let β_i be the angle between ℓ and Π_i , and let β'_i be the angle between ℓ' and Π_i . By Lemma 1 we have $\beta_i, \beta'_i \in O(1/n)$.

Let us choose an orientation on ℓ and ℓ' so that they intersect Π_1 before Π_2 . Since ℓ stabs A before B and C before D , it intersects Π_1 from bottom to top, and Π_2 from left to right. The segment $p_1 p_2$ therefore lies in the top-left quadrant of Figure 3. On the other hand, ℓ' stabs B before A and D before C , so it intersects Π_1 from top to bottom, and Π_2 from right to left, and the segment $p'_1 p'_2$ lies in the bottom-right quadrant of the figure.

Let now $t := d(p_1, p_2)$ and $t' := d(p'_1, p'_2)$. Lemma 3 implies

$$d(g_1, g_2) \leq d(g_1, p_1) + d(p_1, p_2) + d(p_2, g_2) \leq \sin \beta_1 + t + \sin \beta_2 \leq t + O(1/n),$$

and similarly

$$d(g_1, g_2) \leq d(g_1, p'_1) + d(p'_1, p'_2) + d(p'_2, g_2) \leq \sin \beta'_1 + t' + \sin \beta'_2 \leq t' + O(1/n),$$

and so

$$d(g_1, g_2) \leq O(1/n) + \min\{t, t'\}.$$

It remains to prove that $\min\{t, t'\} \leq 1$. Let u_1 (u'_1) be the orthogonal projection of p_1 (p'_1) on Π_2 , u_2 (u'_2) the orthogonal projection of p_2 (p'_2) on Π_1 . Consider the rectangular triangle $p_1 u_2 p_2$. We have $\angle u_2 p_1 p_2 = \beta_1$, and so

$$t \sin \beta_1 = d(p_2, u_2) = d(p_2, \Pi_1). \quad (1)$$

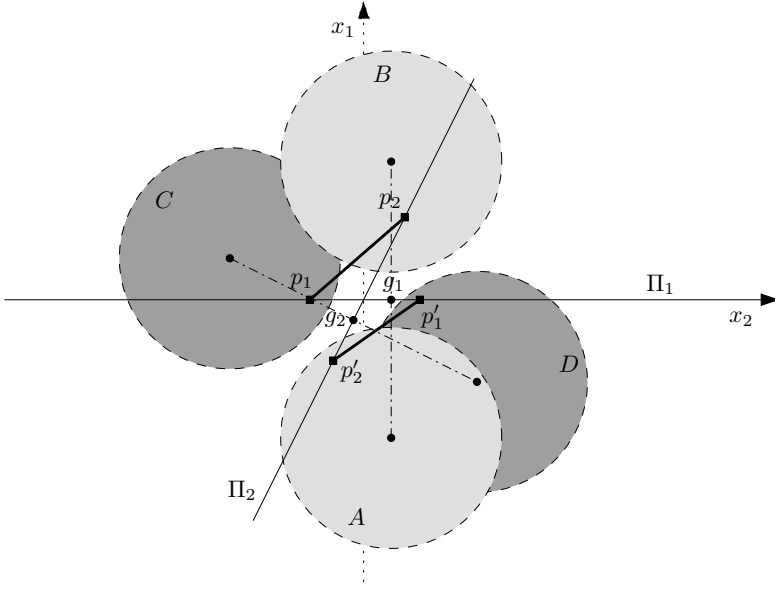


Fig. 3. The two hyperplanes define four quadrants

Similarly, we can consider the rectangular triangles $p_2u_1p_1$, $p'_1u'_2p'_2$, and $p'_2u'_1p'_1$ to obtain

$$t \sin \beta_2 = d(p_1, u_1) = d(p_1, \Pi_2), \quad (2)$$

$$t' \sin \beta'_1 = d(p'_2, u'_2) = d(p'_2, \Pi_1), \quad (3)$$

$$t' \sin \beta'_2 = d(p'_1, u'_1) = d(p'_1, \Pi_2). \quad (4)$$

We now distinguish two cases.

The *first case* occurs if, as in the figure, the x_1 -coordinate of g_2 is ≤ 0 . By Lemma 3 we have $d(p_2, g_2) \leq \sin \beta_2$. Since p_2 and g_2 lie on opposite sides of Π_1 , we have $d(p_2, \Pi_1) \leq \sin \beta_2$. Similarly, we have $d(p_1, g_1) \leq \sin \beta_1$, and p_1 and g_1 lie on opposite sides of Π_2 , implying $d(p_1, \Pi_2) \leq \sin \beta_1$. Plugging into Eq. (1) and (2), we obtain

$$t \leq \min \left\{ \frac{\sin \beta_2}{\sin \beta_1}, \frac{\sin \beta_1}{\sin \beta_2} \right\} \leq 1,$$

which proves the lemma for this case.

The *second case* occurs if the x_1 -coordinate of g_2 is > 0 . We let $s_1 := d(g_1, \Pi_2)$, and $s_2 := d(g_2, \Pi_1)$. Applying Lemma 3, we then have

$$d(p_2, \Pi_1) \leq d(p_2, g_2) + s_2 \leq \sin \beta_2 + s_2, \quad (5)$$

$$d(p_1, \Pi_2) \leq d(p_1, g_1) - s_1 \leq \sin \beta_1 - s_1, \quad (6)$$

$$d(p'_2, \Pi_1) \leq d(p'_2, g_2) - s_2 \leq \sin \beta'_2 - s_2, \quad (7)$$

$$d(p'_1, \Pi_2) \leq d(p'_1, g_1) + s_1 \leq \sin \beta'_1 + s_1. \quad (8)$$

Plugging Ineqs. (5) to (8) into (1) to (4), we obtain

$$t \leq \frac{\sin \beta_2 + s_2}{\sin \beta_1}, \quad (9)$$

$$t \leq \frac{\sin \beta_1 - s_1}{\sin \beta_2}, \quad (10)$$

$$t' \leq \frac{\sin \beta'_2 - s_2}{\sin \beta'_1}, \quad (11)$$

$$t' \leq \frac{\sin \beta'_1 + s_1}{\sin \beta'_2}. \quad (12)$$

We want to prove that $\min(t, t') \leq 1$. We assume the contrary. From $t > 1$ and Ineq. (10) we obtain

$$\sin \beta_2 < \sin \beta_1 - s_1,$$

and from $t' > 1$ and Ineq. (11) we get

$$\sin \beta'_1 < \sin \beta'_2 - s_2.$$

Plugging this into Ineq. (9) and (12) results in

$$\begin{aligned} t &\leq \frac{\sin \beta_2 + s_2}{\sin \beta_1} < \frac{\sin \beta_1 - s_1 + s_2}{\sin \beta_1} = 1 + \frac{s_2 - s_1}{\sin \beta_1}, \\ t' &\leq \frac{\sin \beta'_1 + s_1}{\sin \beta'_2} < \frac{\sin \beta'_2 - s_2 + s_1}{\sin \beta'_2} = 1 + \frac{s_1 - s_2}{\sin \beta'_2}. \end{aligned}$$

It follows that if $s_2 < s_1$ then $t < 1$, otherwise $t' \leq 1$. In either case the lemma follows.

Given a set $\mathcal{S}$ of n spheres, Katchalski et al. [7] define a *switched pair* to be a pair of spheres (A, B) from $\mathcal{S}$ such that there is a line transversal ℓ of $\mathcal{S}$ stabbing A before B and another line transversal ℓ' of $\mathcal{S}$ stabbing B before A . (Both transversals must be oriented in the same direction, as discussed in the remark after Lemma 1.)

The notion of switched pair is well defined because of the following lemma.

Lemma 5. [7, Lemma 2.8] *Let $\mathcal{S}$ be a set of n disjoint unit spheres in $\mathbb{R}^d$, with n large enough. A sphere of $\mathcal{S}$ can appear in at most one switched pair.*

The number of switched pairs determines the number of geometric permutations, as the following lemma shows.

Lemma 6. [7, Lemma 2.9] *Let $\mathcal{S}$ be a set of n disjoint unit spheres in $\mathbb{R}^d$, for n large enough. The two members of a switched pair must appear consecutively in all geometric permutations of $\mathcal{S}$. If there are a total of m switched pairs, then $\mathcal{S}$ admits at most 2^m different geometric permutations.*

The following lemma provides a lower bound on the distance of the centers of gravity of two switched pair. It will be a key ingredient in our proof that only one switched pair can exist, as the lower bound contradicts the upper bound we have shown in Lemma 4.

Lemma 7. [7, Lemma 3.2] *Let $\mathcal{S}$ be a set of n disjoint unit spheres in $\mathbb{R}^d$ with two switched pairs (A, B) and (C, D) . Then*

$$d(g(A, B), g(C, D)) \geq \sqrt{2} - \varepsilon(n),$$

where $\varepsilon(n) > 0$ and $\lim_{n \rightarrow \infty} \varepsilon(n) = 0$.

Finally, the following lemma allows us to apply Lemma 4.

Lemma 8. [7, Lemma 3.1] *Let $\mathcal{S}$ be a set of n disjoint unit spheres in $\mathbb{R}^d$ with two switched pairs (A, B) and (C, D) , for n large enough. Then there are two line transversals ℓ and ℓ' of $\mathcal{S}$ such that ℓ stabs the four spheres in the order $ABCD$ and ℓ' stabs them in the order $BADC$, possibly after interchanging A and B and/or C and D .*

Theorem 1. *A set $\mathcal{S}$ of n disjoint unit spheres in $\mathbb{R}^d$, for n large enough, has at most one switched pair and admits at most two different geometric permutations.*

Proof. The second claim follows from the first by Lemma 6. Assume there are two different switched pairs (A, B) and (C, D) . By Lemma 8 there exist two line transversals ℓ and ℓ' and four spheres A, B, C, D in $\mathcal{S}$ such that ℓ stabs them in the order $ABCD$ and ℓ' stabs them in the order $BADC$. Choosing n large enough, we have by Lemma 7

$$d(g(A, B), g(C, D)) \geq \sqrt{2} - 1/5.$$

By Lemma 4, we also have

$$d(g(A, B), g(C, D)) < 1 + 1/5 < \sqrt{2} - 1/5,$$

a contradiction. The theorem follows.

Acknowledgments. We thank Hervé Brönnimann for introducing the problem to us. Part of this research was done during the Second McGill-INRIA Workshop on Computational Geometry in Computer Graphics at McGill Bellairs Research Institute. We wish to thank the institute for its hospitality, and especially Gwen, Bellair's chef, for her inspiring creations.

References

1. A. Asinowski. Common transversals and geometric permutations. Master's thesis, Technion IIT, Haifa, 1998.
2. H. Edelsbrunner and M. Sharir. The maximum number of ways to stab n convex non-intersecting sets in the plane is $2n - 2$. *Discrete Comput. Geom.*, 5:35–42, 1990.
3. J. E. Goodman, R. Pollack, and R. Wenger. Geometric transversal theory. In J. Pach, editor, *New Trends in Discrete and Computational Geometry. Algorithms and Combinatorics*, vol. 10, pages 163–198. Springer-Verlag, 1993.

4. A. Holmsen, M. Katchalski, and T. Lewis. A Helly-type theorem for line transversals to disjoint unit balls. *Discrete Comput. Geom.*, 29:595–602, 2003.
5. Y. Huang, J. Xu, and D. Z. Chen. Geometric permutations of high dimensional spheres. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 244–245, 2001.
6. M. Katchalski, T. Lewis, and A. Liu. The different ways of stabbing disjoint convex sets. *Discrete Comput. Geom.*, 7:197–206, 1992.
7. M. Katchalski, S. Suri, and Y. Zhou. A constant bound for geometric permutations of disjoint unit balls. *Discrete & Computational Geometry*, 29:161–173, 2003.
8. M. J. Katz and K. R. Varadarajan. A tight bound on the number of geometric permutations of convex fat objects in $\mathbb{R}^d$. *Discrete Comput. Geom.*, 26:543–548, 2001.
9. S. Smorodinsky, J. S. B. Mitchell, and M. Sharir. Sharp bounds on geometric permutations for pairwise disjoint balls in $\mathbb{R}^d$. *Discrete Comput. Geom.*, 23:247–259, 2000.
10. R. Wenger. Upper bounds on geometric permutations for convex sets. *Discrete Comput. Geom.*, 5:27–33, 1990.
11. R. Wenger. Helly-type theorems and geometric transversals. In J. E. Goodman and J. O’Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 4, pages 63–82. CRC Press LLC, Boca Raton, FL, 1997.
12. Y. Zhou and S. Suri. Geometric permutations of balls with bounded size disparity. *Comput. Geom. Theory Appl.*, 26:3–20, 2003.

An Optimal Algorithm for the Maximum-Density Segment Problem

Kai-min Chung¹ and Hsueh-I Lu^{2*}

¹ Dept. Computer Science & Information Engineering, National Taiwan University

² Institute of Information Science, Academia Sinica, 128 Academia Road, Sec. 2, Taipei 115, Taiwan, Republic of China. <http://www.iis.sinica.edu.tw/~hil/>.

hil@iis.sinica.edu.tw.

Abstract. We address a fundamental string problem arising from analysis of biomolecular sequences. The input consists of two integers L and U and a sequence S of n number pairs (a_i, w_i) with $w_i > 0$. Let *segment* $S(i, j)$ of S be the consecutive subsequence of S starting index i to index j . The *density* of $S(i, j)$ is $d(i, j) = (a_i + a_{i+1} + \dots + a_j) / (w_i + w_{i+1} + \dots + w_j)$. The *maximum-density segment problem* is to find a maximum-density segment over all segments of S with $L \leq w_i + w_{i+1} + \dots + w_j \leq U$. The best previously known algorithm for the problem, due to Goldwasser, Kao, and Lu, runs in $O(n \log(U - L + 1))$ time. In the present paper, we solve the problem in $O(n)$ time. Our approach bypasses the complicated *right-skew decomposition*, introduced by Lin, Jiang, and Chao. As a result, our algorithm has the capability to process the input sequence in an online manner, which is an important feature for dealing with genome-scale sequences. Moreover, for an input sequence S representable in $O(k)$ space, we also show how to exploit the sparsity of S and solve the maximum-density segment problem for S in $O(k)$ time.

1 Introduction

We address the following fundamental string problem: The input consists of two integers L and U and a sequence S of number pairs (a_i, w_i) with $w_i > 0$ for $i = 1, \dots, n$. A *segment* $S(i, j)$ is a consecutive subsequence of S starting with index i and ending with index j . For a segment $S(i, j)$, the *width* is $w(i, j) = w_i + w_{i+1} + \dots + w_j$, and the *density* is $d(i, j) = (a_i + a_{i+1} + \dots + a_j) / w(i, j)$. It is not difficult to see that with an $O(n)$ -time preprocessing to compute all $O(n)$ prefix sums $a_1 + a_2 + \dots + a_j$ and $w_1 + w_2 + \dots + w_j$, the density of any segment can be computed in $O(1)$ time. $S(i, j)$ is *feasible* if $L \leq w(i, j) \leq U$. The *maximum-density segment problem* is to find a maximum-density segment over all $O(n^2)$ feasible segments.

This problem arises from the investigation of non-uniformity of nucleotide composition within genomic sequences, which was first revealed through thermal melting and gradient centrifugation experiments [17,23]. The GC content of

* Corresponding author. Research supported in part by NSC grant 91-2215-E-001-001.

the DNA sequences in all organisms varies from 25% to 75%. GC-ratios have the greatest variations among bacteria's DNA sequences, while the typical GC-ratios of mammalian genomes stay in 45-50%. Despite intensive research effort in the past two decades, the underlying causes of the observed heterogeneity remain debatable [3,2,6,8,34,36,9,14,7,4]. Researchers [26,33] observed that the compositional heterogeneity is highly correlated to the GC content of the genomic sequences. Other investigations showed that gene length [5], gene density [38], patterns of codon usage [31], distribution of different classes of repetitive elements [32,5], number of isochores [2], lengths of isochores [26], and recombination rate within chromosomes [10] are all correlated with GC content. More research related to GC-rich segments can be found in [24,25,16,35,29,13,37,12,19] and the references therein.

In the most basic form of the maximum-density segment problem, the sequence S corresponds to the given DNA sequence, where $a_i = 1$ if the corresponding nucleotide in the DNA sequence is G or C; and $a_i = 0$ otherwise. In the work of Huang [15], sequence entries took on values of p and $1 - p$ for some real number $0 \leq p \leq 1$. More generally, we can look for regions where a given set of patterns occur very often. In such applications, a_i could be the relative frequency that the corresponding DNA character appears in the given patterns. Further natural applications of this problem can be designed for sophisticated sequence analysis such as mismatch density [30], ungapped local alignments [1], annotated multiple sequence alignments [33], promoter mapping [18], and promoter recognition [27].

For the *uniform* case, i.e., $w_i = 1$ for all indices i , Nekrutendo and Li [26], and Rice, Longden and Bleasby [28] employed algorithms for the case $L = U$, which is trivially solvable in $O(n)$ time. More generally, when $L \neq U$, the problem is also easily solvable in $O(n(U - L + 1))$ time, linear in the number of feasible segments. Huang [15] studied the case where $U = n$, i.e., there is effectively no upper bound on the width of the desired maximum-density segments. He observed that an optimal segment exists with width at most $2L - 1$. Therefore, this case is equivalent to the case with $U = 2L - 1$ and can be solved in $O(nL)$ time in a straightforward manner. Lin, Jiang, and Chao [22] gave an $O(n \log L)$ -time algorithm for this case based on right-skew decompositions of a sequence. (See [21] for a related software.) The case with general U was first investigated by Goldwasser, Kao, and Lu [11], who gave an $O(n)$ -time algorithm for the uniform case. (Recently, Kim [20] showed an alternative algorithm based upon an interesting geometric interpretation of the problem. Unfortunately, the analysis of time complexity has some flaw which seems hard to fix.¹) For the general (i.e.,

¹ Kim claims that all the progressive updates of the lower convex hulls $L_j \cup R_j$ can be done in linear time. The paper only sketches how to obtain $L_{j+1} \cup R_{j+1}$ from $L_j \cup R_j$. (See the fourth-to-last paragraph of page 340 in [20].) Unfortunately, Kim seems to overlook the marginal cases when the upper bound U forces the p_z of $L_j \cup R_j$ to be deleted from $L_{j+1} \cup R_{j+1}$. As a result, obtaining $L_{j+1} \cup R_{j+1}$ from $L_j \cup R_j$ could be much more complicated than Kim's sketch. We believe that any correct implementation of Kim's algorithm may require $\Omega(n \log(U - L + 1))$ time.

```

algorithm MAIN
1   let  $i_{j_0-1} = 1$ ;
2   for  $j = j_0$  to  $n$  do
3       output  $i_j = \text{FIND}(\max(i_{j-1}, \ell_j), j)$ ;

subroutine FIND( $x, j$ )
1   let  $i = x$ ;
2   while  $i < r_j$  and  $d(i, \phi(i, r_j - 1)) \leq d(i, j)$  do
3       let  $i = \phi(i, r_j - 1) + 1$ ;
4   return  $i$ ;

```

Fig. 1. The framework of our algorithm.

non-uniform) case, Goldwasser, Kao, and Lu [11] also gave an $O(n \log(U - L + 1))$ -time algorithm. By bypassing the complicated preprocessing step required in [11], We successfully reduce the required time for the general case down to $O(n)$. Our result is based upon the following equations, stating that the order of $d(x, y)$, $d(y + 1, z)$, and $d(x, z)$ with $x \leq y < z$ can be determined by that of any two of them:

$$d(x, y) \leq d(y + 1, z) \Leftrightarrow d(x, y) \leq d(x, z) \Leftrightarrow d(x, z) \leq d(y + 1, z); \quad (1)$$

$$d(x, y) < d(y + 1, z) \Leftrightarrow d(x, y) < d(x, z) \Leftrightarrow d(x, z) < d(y + 1, z). \quad (2)$$

(Both equations can be easily verified by observing the existence of some number ρ with $0 < \rho < 1$ and $d(x, z) = d(x, y)\rho + d(y + 1, z)(1 - \rho)$.) Our algorithm is capable of processing the input sequence in an online manner, which is an important feature for dealing with genome-scale sequences.

For bioinformatics applications in [30,1,33,18,27], the input sequence S is usually very *sparse*, e.g., S can be represented by k triples (a_i, w_i, n_i) to signify that all entries of $S(n_1 + n_2 + \dots + n_{i-1} + 1, n_1 + n_2 + \dots + n_i)$ are (a_i, w_i) for $i = 1, 2, \dots, k$. In this paper we also show how to exploit the sparsity of S and solve the maximum-density problem for S given in the above compact representation in $O(k)$ time.

The remainder of the paper is organized as follows. Section 2 shows the main algorithm. Section 3 explains how to cope with the simple case that the width upper bound U is ineffective. Section 4 takes care of the more complicated case that U is effective. Section 5 explains how to exploit the sparsity of the input sequence.

2 The Main Algorithm

For any integers x and y , let $[x, y]$ denote the set $\{x, x + 1, \dots, y\}$. Throughout the paper, we need the following definitions and notation with respect to the input length- n sequence S and width bounds L and U . Let j_0 be the smallest

index with $w(1, j_0) \geq L$. Let $J = [j_0, n]$. For each $j \in J$, let ℓ_j (respectively, r_j) be the smallest (respectively, largest) index i with $L \leq w(i, j) \leq U$. That is, $S(i, j)$ is feasible if and only if $i \in [\ell_j, r_j]$. Clearly, for the uniform case, we have $\ell_{i+1} = \ell_i + 1$ and $r_{i+1} = r_i + 1$. As for the general case, we only know that ℓ_j and r_j are both (not necessarily monotonically) increasing. One can easily compute all ℓ_j and r_j in $O(n)$ time. Let i_j^* be the largest index $k \in [\ell_j, r_j]$ with $d(k, j) = \max\{d(i, j) : i \in [\ell_j, r_j]\}$. Clearly, there must be an index j such that $S(i_j^*, j)$ is a maximum-density segment of S . Therefore, a natural (but seemingly difficult) possibility to optimally solve the maximum-density segment problem would be to compute i_j^* for all indices $j \in J$ in $O(n)$ time. Define $\phi(x, y)$ to be the largest index $k \in [x, y]$ with $d(x, k) = \min\{d(i, i), d(i, i+1), \dots, d(i, j)\}$. That is, $S(x, \phi(x, y))$ is the longest minimum-density prefix of $S(x, y)$. Our strategy is to compute an index $i_j \in [\ell_j, r_j]$ for each index $j \in J$ by the algorithm shown in Figure 1. The following lemma ensures the correctness of our algorithm, and thus reduces the maximum-density segment problem to implementing the algorithm to run in $O(n)$ time.

Lemma 1. $\max_{j \in J} d(i_j, j) = \max_{j \in J} d(i_j^*, j).$

Proof. Let t be an index in J with $d(i_t^*, t) = \max_{j \in J} d(i_j^*, j)$. Clearly, it suffices to show $i_t = i_t^*$. If $i_t < i_t^*$, then $i_t < r_t$. By $d(i_t, t) \leq d(i_t^*, t)$ and Equation (1), we have $d(i_t, i_t^* - 1) \leq d(i_t, t)$. By $i_t^* - 1 \leq r_t - 1$, we have $d(i_t, \phi(i_t, r_t - 1)) \leq d(i_t, t)$, contradicting the definitions of FIND and i_t . To prove $i_t \leq i_t^*$, we assume $s \leq t$ for contradiction, where s is the smallest index in J with $i_s > i_t^*$. By $s \leq t$, we know $\ell_s \leq i_t^*$. By definition of FIND and $i_{s-1} \leq i_t^*$, there is an index $i \in J$ with $\max(i_{s-1}, \ell_s) \leq i \leq i_t^* \leq k < i_s$ and $d(i, k) \leq d(i, s)$, where $k = \phi(i, r_s - 1)$. By $i_t^* \leq k < i_s$ and $s \leq t$, we know $\ell_t \leq k + 1 \leq r_t$. By definition of i_t^* and $i_t^* < k + 1$, we have $d(k + 1, t) < d(i_t^*, t)$, which by Equation (2) implies $d(i_t^*, t) < d(i_t^*, k)$. By $k = \phi(i, r_s - 1)$ and Equation (1), we know $d(i_t^*, k) \leq d(i, k)$ by observing that $i < i_t^*$ implies $d(i, i_t^* - 1) \leq d(i, k)$. Thus, we have $d(i_t^*, t) < d(i, s)$, contradicting the definitions of t and i_t^* . $\square$

One can verify that the value of i increases by at least one each time Step 3 of FIND is executed. Therefore, to implement the algorithm to run in $O(n)$ time, it suffices to maintain a data structure to support $O(1)$ -time query for each $\phi(i, r_j - 1)$ in Step 2 of FIND.

3 Coping with Ineffective Width Upper Bound

When U is ineffective, i.e., $U \geq w(1, n)$, we have $\ell_j = 1$ for all $j \in F$. Therefore, the function call in Step 3 of MAIN is exactly $\text{FIND}(i_{j-1}, j)$. Moreover, during the execution of the function call $\text{FIND}(i_{j-1}, j)$, the value of i can only be i_{j-1} , $\phi(i_{j-1}, r_j - 1) + 1$, $\phi(\phi(i_{j-1}, r_j - 1) + 1, r_j - 1) + 1$, $\dots$, etc. Suppose that a subroutine call to $\text{UPDATE}(j)$ yields an array Φ of indices and two indices p and q of Φ with $p \leq q$ such that the following condition C_j holds:

$$- \Phi[p] = i_{j-1},$$

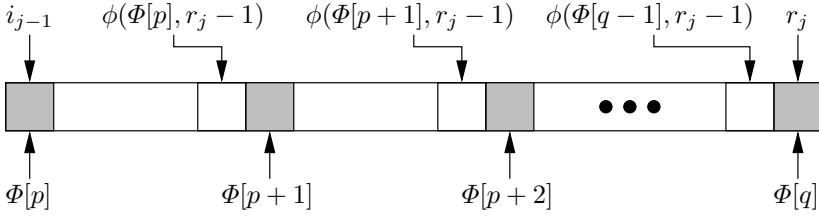


Fig. 2. An illustration for condition C_j .

```

subroutine FIND( $j$ )
1   UPDATE( $j$ );
2   while  $p < q$  and  $d(\Phi[p], \Phi[p+1] - 1) \leq d(\Phi[p], j)$  do
3       let  $p = p + 1$ ;
4   return  $\Phi[p]$ ;

subroutine UPDATE( $j$ )
1   for  $k = r_{j-1} + 1$  to  $r_j$  do {
2       while  $p < q$  and  $d(\Phi[q-1], \Phi[q] - 1) \geq d(\Phi[q-1], k - 1)$  do
3           let  $q = q - 1$ ;
4       let  $q = q + 1$ ;
5       let  $\Phi[q] = k$ ;
6   }

```

Fig. 3. The implementation for the case that U is ineffective.

- $\Phi[q] = r_j$, and
- $\Phi[t] = \phi(\Phi[t-1], r_j - 1) + 1$ holds for each index $t \in [p+1, q]$.

See Figure 2 for an illustration. Then, the subroutine call to $\text{FIND}(i_{j-1}, j)$ can clearly be replaced by $\text{FIND}(j)$, as defined in Figure 3. That is, one can look up the value of $\phi(i, r_j - 1)$ from Φ in $O(1)$ time. It remains to show how to implement $\text{UPDATE}(j)$ such that all of its $O(n)$ subroutine calls together run in $O(n)$ time. Initially, we assign $p = 1$ and $q = 0$. Let subroutine UPDATE be as shown in Figure 3. The following lemmas ensure the correctness of our implementation.

Lemma 2. *For each $j \in J$, condition C_j holds right after we finish the subroutine call to $\text{UPDATE}(j)$.*

Proof. It is not difficult to verify that with the initialization $p = 1$ and $q = 0$, condition C_{j_0} holds with $p = 1$ and $q \geq 1$ after calling $\text{UPDATE}(j_0)$.

Now consider the moment when we are about to make a subroutine call $\text{UPDATE}(j)$ for an index $j \in J - \{j_0\}$. Since the subroutine call to $\text{FIND}(i_{j-2}, j-1)$ was just finished, we have that $\Phi[p] = i_{j-1}$, $\Phi[q] = r_{j-1}$, and $\Phi[t] = \phi(\Phi[t-1], r_{j-1} - 1) + 1$ holds for each index $t \in [p+1, q]$. Observe that $\phi(y, k-1)$ is either $\phi(y, k-2)$ or $k-1$. Moreover, $\phi(y, k-1) = k-1$ if and only if

$d(y, \phi(y, k - 2)) \geq d(y, k - 1)$. Therefore, one can verify that at the end of each iteration of the for-loop of $\text{UPDATE}(j)$, we have that $\Phi[p] = i_{j-1}$, $\Phi[q] = k$, and $\Phi[t] = \phi(\Phi[t - 1], k - 1) + 1$ holds for each index $t \in [p + 1, q]$. (The value of q may change, though.) It follows that at the end of the for-loop, condition C_j holds. $\square$

Lemma 3. *The implementation shown in Figure 3 runs in $O(n)$ time.*

Proof. Observe that each iteration of the while-loops of FIND and UPDATE decreases the value of $q - p$ by one. Since Step 4 of UPDATE runs $O(n)$ times, the lemma can be proved by verifying that $q - p \geq -1$ holds throughout the execution of MAIN . $\square$

By Lemmas 2 and 3, we have an $O(n)$ -time algorithm for the case with ineffective width upper bound.

4 Coping with Effective Width Upper Bound

In contrast to the previous simple case, when U is arbitrary, ℓ_j may not always be 1. Therefore, the first argument of the function call in Step 3 of MAIN could be ℓ_j with $\ell_j > i_{j-1}$. It seems quite difficult to update the corresponding data structure Φ in overall linear time such that condition C_j holds throughout the execution of our algorithm. To overcome the difficulty, our algorithm maintains an alternative (weaker) condition. As a result, the located index in the t -th iteration could be larger than i_t^* , where t is an index in J with $d(i_t^*, t) = \max_{j \in J} d(i_j^*, j)$. Fortunately, this potential problem can be resolved if we simultaneously solve a variant version of the maximum-density segment problem. The details follow.

4.1 A Variant Version of the Maximum-Density Segment Problem

Suppose that we are given an index interval $X = [x_1, x_2]$. Let $Y = [y_1, y_2]$ be the interval such that y_1 is the smallest index with $w(x_2, y_1) \geq L$, and y_2 is the largest index with $w(x_2, y_2) \leq U$. The *variant version* of the maximum-density segment problem is to look for indices i and j with $i \in X$, $j \in Y$, and $w(i, j) \leq U$ such that $d(i, j)$ is maximized, i.e.,

$$d(i, j) = \max_{x \in Y, y \in Y, w(x, y) \leq U} d(x, y).$$

For each $j \in Y$, let k_j^* be the largest index $x \in X$ with $L \leq w(x, j) \leq U$ that maximizes $d(x, j)$. Although solving the variant version can naturally be reduced to computing the index k_j^* for each index $j \in Y$, the required running time will be more than what we can afford. Instead, we compute an index k_j for each index $j \in J$ such that the following lemma holds

Lemma 4. $\max_{j \in J} d(k_j, j) = \max_{j \in J} d(k_j^*, j)$.

```

algorithm VARIANT( $x_1, x_2$ )
1   let  $y_1$  be the smallest index with  $w(x_2, y_1) \geq L$ ;
2   let  $y_2$  be the smallest index with  $w(x_2, y_2) \leq U$ ;
3   let  $k_{y_1-1} = x_1$ ;
4   for  $j = y_1$  to  $y_2$  do
5       output  $k_j = \text{VFIND}(\max(k_{j-1}, \ell_j), j)$ ;

subroutine VFIND( $x, j$ )
1   let  $i = x$ ;
2   while  $i < x_2$  and  $d(i, \phi(i, x_2 - 1)) \leq d(i, j)$  do
3       let  $i = \phi(i, x_2 - 1) + 1$ ;
4   return  $i$ ;

```

Fig. 4. Our algorithm for the variant version of the maximum-density segment problem.

By $w(x_2, y_1) \geq L$ and $w(x_2, y_2) \leq U$, one can easily see that x_2 is always the largest index $x \in X$ with $L \leq w(x, j) \leq U$. Our algorithm for solving the problem is as shown in Figure 4, which is presented in a way to emphasize the analogy to the algorithm shown in Figure 1. For example, the index k_j in Figure 4 is the counterpart of the index i_j in Figure 1. Also, the indices x_1 and x_2 in Figure 4 play the role of the indices ℓ_j and r_j in Figure 1. Lemma 4 can be proved in a way very similar to the proof of Lemma 1.

Again, the challenge lies supporting the $O(1)$ -time query for $\phi(x, y)$. Fortunately, unlike in algorithm MAIN, where both parameters x and y are changed during the execution, the second parameter y is fixed to $x_2 - 1$. Therefore, to support each query to $\phi(i, x_2 - 1)$ in $O(1)$ time, we can actually afford to spend $O(x_2 - x_1)$ time to compute a data structure Ψ such that $\Phi[i] = \phi(i, x_2 - 1)$ for each $i \in [x_1, x_2 - 1]$. Specifically, the subroutine VFIND can be implemented as shown in Figure 5. We have the following lemma.

Lemma 5. *The implementation shown in Figure 5 solves the variant version of the maximum-density segment problem in $O(x_2 - x_1 + y_2 - y_1 + 1)$ time.*

Proof. (sketch) The correctness of the implementation can be proved by verifying that if $\Psi[z] = \phi(z, y)$ holds for each $z = p + 1, p + 2, \dots, y$, then $\phi(p, y)$ has to be in the set $\{p, \Psi[p + 1], \Psi[\Psi[p + 1] + 1], \dots, y\}$.

One can see that the running time is indeed $O(x_2 - x_1 + y_2 - y_1 + 1)$ by verifying that throughout the execution of the implementation, (a) the while-loop of vfind runs $O(y_2 - y_1 + 1)$ iterations, and (b) the while-loop of VPREPARE runs $O(x_2 - x_1 + 1)$ iterations. To see statement (a), just observe that the value of index i (i) never decreases, (ii) stays in $[x_1, x_2]$, and (iii) increases by at least one each time Step 3 of VFIND is executed. As for statement (b), let Λ_p denote the cardinality of the set $\{p, \Psi[p + 1], \Psi[\Psi[p + 1] + 1], \dots, y\}$. Consider the iteration with index p of the for-loop of VPREPARE. Note that if Step 6 of VPREPARE executes t_p times in this iteration, then we have $\Lambda_p = \Lambda_{p+1} - t_p + 1$. Since

```

algorithm VARIANT( $x_1, x_2$ )
1   let  $y_1$  be the smallest index with  $w(x_2, y_1) \geq L$ ;
2   let  $y_2$  be the smallest index with  $w(x_2, y_1) \leq U$ ;
3   VPREPARE( $x_1, x_2 - 1$ );
4   let  $k_{y_1-1} = x_1$ ;
5   for  $j = y_1$  to  $y_2$  do
6     output  $k_j = \text{VFIND}(\max(k_{j-1}, \ell_j), j)$ ;

subroutine VFIND( $x, j$ )
1   let  $i = x$ ;
2   while  $i < x_2$  and  $d(i, \Psi[i]) \leq d(i, j)$  do
3     let  $i = \Psi[i] + 1$ ;
4   return  $i$ ;

subroutine VPREPARE( $x, y$ )
1   let  $\Psi[y] = y$ ;
2   for  $p = y - 1$  downto  $x$  do
3     let  $q = p$ ;
4     while  $d(p, q) \geq d(p, \Psi[q + 1])$  and  $\Psi[q + 1] < y$  do
5       let  $q = \Psi[q + 1]$ ;
6     let  $\Psi[p] = q$ ;

```

Fig. 5. The implementation for the variant version.

$A_p \geq 1$ holds for each $p \in X$, we have $\sum_{p \in X} t_p = O(x_2 - x_1 + 1)$, and thus statement (b) holds. $\square$

4.2 Our Algorithm for the General Case

With the help the linear-time algorithm for solving the variant version shown in the previous subsection, we can construct a linear-time algorithm for solving the original maximum-density segment problem by slightly modify Step 3 of MAIN as follows.

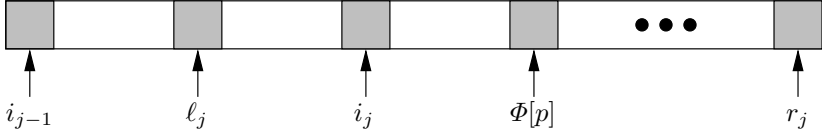
- If $i_{j-1} \geq \ell_j$, the subroutine call $\text{FIND}(\max(i_{j-1}, \ell_j), j)$ can be replaced by $\text{FIND}(j)$ as explained in Section 3.
- If $i_{j-1} < \ell_j$, we cannot afford to appropriately update the data structure Φ . For this case, instead of moving the head i to ℓ_j , we move i to $\Phi[p]$, where p is the smallest index with $\ell_j \leq \Phi[p]$. the first element of $\phi[i, r_j - 1]$ which is on the right-hand side of ℓ_j .
Of course, when we assign $\Phi[p]$ to i , we may overlook the possibilities of i_j being in the interval $[i_{j-1}, \Phi[p] - 1]$. (See the illustration shown in Figure 7.) This is when the variant version comes in: it turns out that we can remedy the potential problem by calling $\text{VARIANT}(i_{j-1}, \Phi[p] - 1)$.

The algorithm for solving the general case is shown in Figure 6.

algorithm GENERAL

```

1  let  $i_{j_0-1} = 1$ ,  $p = 1$ , and  $q = 0$ ;
2  for  $j = j_0$  to  $n$  do
3    while  $\Phi[p] < \ell_j$  do
4      let  $p = p + 1$ ;
5    if  $i_{j-1} < \Phi[p]$  then
6      call VARIANT( $i_{j-1}, \Phi[p] - 1$ );
7    output  $i_j = \text{FIND}(j)$ ;
```

Fig. 6. Our algorithm for the general case.**Fig. 7.** Illustration for the situation when Step 6 of GENERAL is required.

One can see the correctness of the algorithm by verifying that $i_t^* \in \{i_t, k_t\}$ holds for any index t that maximizes $d(i_t^*, t)$. By the explanation of Section 3, we know that those subroutine calls performed in Step 7 of GENERAL runs in overall $O(n)$ time. To see that the algorithm GENERAL indeed runs in $O(n)$ time, it suffices to prove that all subroutine calls to VARIANT in Step 6 take $O(n)$ time in total. Suppose that $x_{s,1}$ and $x_{s,2}$ are the arguments for the s -th subroutine call to VARIANT in Step 6 of GENERAL. One can easily verify that the intervals $[x_{s,1}, x_{s,2}]$ are mutually disjoint for all indices s . It follows that all those corresponding intervals $[y_{s,1}, y_{s,2}]$ for the index j are also mutually disjoint. By Lemma 5, the overall running time is $\sum_s O(x_{s,2} - x_{s,1} + y_{s,2} - y_{s,1} + 1) = O(n)$. It is also not difficult to see that our algorithm shown in Figure 6 is already capable of processing the input sequence in an online manner, since our approach requires no preprocessing at all. We summarize our main result in the following theorem.

Theorem 1. *Given two width bounds L and U and a length- n sequence S , a maximum-density feasible segment of S can be found in $O(n)$ time in an online manner.*

5 Exploiting Sparsity

Suppose that the input sequence is given in a compact representation consisting of k triples (a_i, w_i, n_i) to specify that all entries of $S(n_1 + n_2 + \dots + n_{i-1} + 1, n_1 + n_2 + \dots + n_i)$ are (a_i, w_i) for $i = 1, 2, \dots, k$. We conclude the paper by showing how to exploit the sparsity of S and solve the maximum-density problem for S in $O(k)$ time based upon the following lemma.

Lemma 6. Let $S(p, q)$ be the maximum-density segment of S . If $(a_p, w_p) = (a_{p-1}, w_{p-1})$ and $d(p, q) \neq d(p, p)$, then either $w(p-1, q) > U$ or $w(p+1, q) < L$. Similarly, if $(a_q, w_q) = (a_{q+1}, w_{q+1})$ and $d(p, q) \neq d(q, q)$, then either $w(p, q+1) > U$ or $w(p, q-1) < L$.

Proof. Assume that $w(p-1, q) \leq U$. By Equation (2), the optimality of $S(p, q)$, and $d(p, q) \neq d(p, p)$, we have $d(p, p) = d(p-1, p-1) < d(p-1, q) < d(p, q)$, and thus $d(p, p) < d(p, q) < d(p+1, q)$. Since $S(p, q)$ is the maximum-density segment, $S(p+1, q)$ has to be infeasible, i.e., $w(p+1, q) < L$. The second statement can be proved similarly. $\square$

We call each (a_i, w_i, n_i) with $1 \leq i \leq k$ a *piece* of S . Lemma 6 states that a segment with head or tail inside a piece can be a maximum-density segment only if its width is close to L or U . Since there are only $O(k)$ such candidates. Therefore, by enumerating all such candidates and running our algorithm with input $(n_1 a_1, n_1 w_1), \dots, (n_k a_k, n_k w_k)$, one can easily modify our algorithm stated in Theorem 1 to run in $O(k)$ time.

Acknowledgments. We thank Yi-Hsuan Hsin and Hsu-Cheng Tsai for discussions in the preliminary stage of this research. We also thank the anonymous reviewers for their helpful comments which significantly improve the presentation of our paper.

References

1. N. N. Alexandrov and V. V. Solovyev. Statistical significance of ungapped sequence alignments. In *Proceedings of Pacific Symposium on Biocomputing*, volume 3, pages 461–470, 1998.
2. G. Barhadi. Isochores and the evolutionary genomics of vertebrates. *Gene*, 241:3–17, 2000.
3. G. Bernardi and G. Bernardi. Compositional constraints and genome evolution. *Journal of Molecular Evolution*, 24:1–11, 1986.
4. B. Charlesworth. Genetic recombination: patterns in the genome. *Current Biology*, 4:182–184, 1994.
5. L. Duret, D. Mouchiroud, and C. Gautier. Statistical analysis of vertebrate sequences reveals that long genes are scarce in GC-rich isochores. *Journal of Molecular Evolution*, 40:308–371, 1995.
6. A. Eyre-Walker. Evidence that both G+C rich and G+C poor isochores are replicated early and late in the cell cycle. *Nucleic Acids Research*, 20:1497–1501, 1992.
7. A. Eyre-Walker. Recombination and mammalian genome evolution. *Proceedings of the Royal Society of London Series B, Biological Science*, 252:237–243, 1993.
8. J. Filipski. Correlation between molecular clock ticking, codon usage fidelity of DNA repair, chromosome banding and chromatin compactness in germline cells. *FEBS Letters*, 217:184–186, 1987.
9. M. P. Francino and H. Ochman. Isochores result from mutation not selection. *Nature*, 400:30–31, 1999.
10. S. M. Fullerton, A. B. Carvalho, and A. G. Clark. Local rates of recombination are positively correlated with GC content in the human genome. *Molecular Biology and Evolution*, 18(6):1139–1142, 2001.

11. M. H. Goldwasser, M.-Y. Kao, and H.-I. Lu. Fast algorithms for finding maximum-density segments of a sequence with applications to bioinformatics. In R. Guigó and D. Gusfield, editors, *Proceedings of the Second International Workshop of Algorithms in Bioinformatics*, Lecture Notes in Computer Science 2452, pages 157–171, Rome, Italy, 2002. Springer.
12. P. Guldberg, K. Gronbak, A. Aggerholm, A. Platz, P. thor Straten, V. Ahrenkiel, P. Hokland, and J. Zeuthen. Detection of mutations in GC-rich DNA by bisulphite denaturing gradient gel electrophoresis. *Nucleic Acids Research*, 26(6):1548–1549, 1998.
13. W. Henke, K. Herdel, K. Jung, D. Schnorr, and S. A. Loening. Betaine improves the PCR amplification of GC-rich DNA sequences. *Nucleic Acids Research*, 25(19):3957–3958, 1997.
14. G. P. Holmquist. Chromosome bands, their chromatin flavors, and their functional features. *American Journal of Human Genetics*, 51:17–37, 1992.
15. X. Huang. An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. *Computer Applications in the Biosciences*, 10(3):219–225, 1994.
16. K. Ikehara, F. Amada, S. Yoshida, Y. Mikata, and A. Tanaka. A possible origin of newly-born bacterial genes: significance of GC-rich nonstop frame on antisense strand. *Nucleic Acids Research*, 24(21):4249–4255, 1996.
17. R. B. Inman. A denaturation map of the λ phage DNA molecule determined by electron microscopy. *Journal of Molecular Biology*, 18:464–476, 1966.
18. I. P. Ioshikhes and M. Q. Zhang. Large-scale human promoter mapping using CpG islands. *Nature Genetics*, 26:61–63, 2000.
19. R. Jin, M.-E. Fernandez-Beros, and R. P. Novick. Why is the initiation nick site of an AT-rich rolling circle plasmid at the tip of a GC-rich cruciform? *The EMBO Journal*, 16(14):4456–4466, 1997.
20. S. K. Kim. Linear-time algorithm for finding a maximum-density segment of a sequence. *Information Processing Letters*, 86(6):339–342, 2003.
21. Y.-L. Lin, X. Huang, T. Jiang, and K.-M. Chao. MAVG: locating non-overlapping maximum average segments in a given sequence. *Bioinformatics*, 19(1):151–152, 2003.
22. Y.-L. Lin, T. Jiang, and K.-M. Chao. Algorithms for locating the length-constrained heaviest segments, with applications to biomolecular sequence analysis. *Journal of Computer and System Sciences*, 65(3):570–586, 2002.
23. G. Macaya, J.-P. Thiery, and G. Bernardi. An approach to the organization of eukaryotic genomes at a macromolecular level. *Journal of Molecular Biology*, 108:237–254, 1976.
24. C. S. Madsen, C. P. Regan, and G. K. Owens. Interaction of CARG elements and a GC-rich repressor element in transcriptional regulation of the smooth muscle myosin heavy chain gene in vascular smooth muscle cells. *Journal of Biological Chemistry*, 272(47):29842–29851, 1997.
25. S.-i. Murata, P. Herman, and J. R. Lakowicz. Texture analysis of fluorescence lifetime images of AT- and GC-rich regions in nuclei. *Journal of Hystochemistry and Cytochemistry*, 49:1443–1452, 2001.
26. A. Nekrutenko and W.-H. Li. Assessment of compositional heterogeneity within and between eukaryotic genomes. *Genome Research*, 10:1986–1995, 2000.
27. U. Ohler, H. Niemann, G. Liao, and G. M. Rubin. Joint modeling of DNA sequence and physical properties to improve eukaryotic promoter recognition. *Bioinformatics*, 17(S1):S199–S206, 2001.

28. P. Rice, I. Longden, and A. Bleasby. EMBOSS: The European molecular biology open software suite. *Trends in Genetics*, 16(6):276–277, June 2000.
29. L. Scotto and R. K. Assoian. A GC-rich domain with bifunctional effects on mRNA and protein levels: implications for control of transforming growth factor beta 1 expression. *Molecular and Cellular Biology*, 13(6):3588–3597, 1993.
30. P. H. Sellers. Pattern recognition in genetic sequences by mismatch density. *Bulletin of Mathematical Biology*, 46(4):501–514, 1984.
31. P. M. Sharp, M. Averof, A. T. Lloyd, G. Matassi, and J. F. Peden. DNA sequence evolution: the sounds of silence. *Philosophical Transactions of the Royal Society of London Series B, Biological Sciences*, 349:241–247, 1995.
32. P. Soriano, M. Meunier-Rotival, and G. Bernardi. The distribution of interspersed repeats is nonuniform and conserved in the mouse and human genomes. *Proceedings of the National Academy of Sciences of the United States of America*, 80:1816–1820, 1983.
33. N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller, and R. Hardison. Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. *Nucleic Acids Research*, 27:3899–3910, 1999.
34. N. Sueoka. Directional mutation pressure and neutral molecular evolution. *Proceedings of the National Academy of Sciences of the United States of America*, 80:1816–1820, 1988.
35. Z. Wang, E. Lazarov, M. O'Donnel, and M. F. Goodman. Resolving a fidelity paradox: Why *Escherichia coli* DNA polymerase II makes more base substitution errors in at- compared to GC-rich DNA. *Journal of Biological Chemistry*, 277:4446–4454, 2002.
36. K. H. Wolfe, P. M. Sharp, and W.-H. Li. Mutation rates differ among regions of the mammalian genome. *Nature*, 337:283–285, 1989.
37. Y. Wu, R. P. Stulp, P. Elfferich, J. Osinga, C. H. Buys, and R. M. Hofstra. Improved mutation detection in GC-rich DNA fragments by combined DGGE and CDGE. *Nucleic Acids Research*, 27(15):e9, 1999.
38. S. Zoubak, O. Clay, and G. Bernardi. The gene distribution of the human genome. *Gene*, 174:95–102, 1996.

Estimating Dominance Norms of Multiple Data Streams

Graham Cormode^{1*} and S. Muthukrishnan^{2**}

¹ Center for Discrete Mathematics and Computer Science, Rutgers University, New Jersey USA, graham@dimacs.rutgers.edu.

² Division of Computer Science, Rutgers University, New Jersey USA, muthu@cs.rutgers.edu and AT&T Research.

Abstract. There is much focus in the algorithms and database communities on designing tools to manage and mine data streams. Typically, data streams consist of multiple signals. Formally, a stream of multiple signals is $(i, a_{i,j})$ where i 's correspond to the domain, j 's index the different signals and $a_{i,j} \geq 0$ give the value of the j th signal at point i . We study the problem of finding norms that are cumulative of the multiple signals in the data stream.

For example, consider the max-dominance norm, defined as $\sum_i \max_j \{a_{i,j}\}$. It may be thought as estimating the norm of the “upper envelope” of the multiple signals, or alternatively, as estimating the norm of the “marginal” distribution of tabular data streams. It is used in applications to estimate the “worst case influence” of multiple processes, for example in IP traffic analysis, electrical grid monitoring and financial domain. In addition, it is a natural measure, generalizing the union of data streams or counting distinct elements in data streams.

We present the first known data stream algorithms for estimating max-dominance of multiple signals. In particular, we use workspace and time-per-item that are both sublinear (in fact, poly-logarithmic) in the input size. In contrast other notions of dominance on streams a, b — min-dominance ($\sum_i \min_j \{a_{i,j}\}$), count-dominance ($|\{i | a_i > b_i\}|$) or relative-dominance ($\sum_i a_i / \max\{1, b_i\}$) — are all impossible to estimate accurately with sublinear space.

1 Introduction

Data streams are emerging as a powerful, new data source. Data streams comprise data generated rapidly over time in massive amounts; each data item must be processed quickly as it is generated. Data streams arise in monitoring telecommunication networks, sensor observations, financial transactions, etc. A significant scenario — and our motivating application — arises in IP networking where Internet Service Providers (ISPs) monitor (a) logs of total number of bytes or packets sent per minute per link connecting the routers in the network, or (b) logs of IP “flow” which are roughly distinct IP sessions characterized by source and destination IP addresses, source and destination port numbers etc. on each link, or at a higher level (c) logs of web clicks and so on. Typically, the logs are monitored in near-real time for simple indicators of “actionable” events, such as anomalies, large concurrence of faults, “hot spots”, and surges, as part

* Supported by NSF ITR 0220280 and NSF EIA 02-05116.

** Supported by NSF CCR 0087022, NSF ITR 0220280 and NSF EIA 02-05116.

of the standard operations of ISPs. Systems in such applications need flexible methods to define, monitor and mine such indicators in data streams.

The starting point of our investigation here is the observation that data streams are often not individual signals, but they comprise multiple signals presented in an interspersed and distributed manner. For example, web click streams may be arbitrary ordering of clicks by different customers at different web servers of a server farm; financial events may be stock activity from multiple customers on stocks from many different sectors and indices; and IP traffic logs may be logs at management stations of the cumulative traffic at different time periods from multiple router links. Even at a single router, there are several interfaces, each of which has multiple logs of traffic data. Our focus here from a conceptual point of view is on suitable norms to measure and monitor about the set of all distributions we see in the cumulative data stream.

Previous work on norm estimation and related problems on data streams has been extensive, but primarily focused on *individual* distributions. For the case of multiple distributions, prior work has typically focused on processing each distribution *individually* so that multiple distributions can be compared based on estimating pairwise distances such as L_p norms [11,17]. These L_p norms are linear so that the per-distribution processing methods can be used to index, cluster multiple distributions or do proximity searches; however, all such methods involve storing space proportional to the number of distinct distributions in the data stream. As such, they do not provide a mechanism to directly understand trends in multiple distributions.

Our motivating scenario is one of a rather large number of distributions as in the IP network application above. In particular, we initiate the study of norms for cumulative trends in presence of multiple distributions. For some norms of this type (in particular, the max-dominance norm to be defined soon), we present efficient algorithms in the data stream model that use space independent of the number of distributions in the signal. For a few other norms, we show hardness results. In what follows, we provide details on the data stream model, on dominance norms and our results.

1.1 Data Stream Model

The model here is that the data stream is a series of items $(i, a_{i,j})$ presented in some *arbitrary order*; i 's correspond to the domain of the distributions (assumed to be identical without loss of generality), j 's to the different distributions and $a_{i,j}$ is the value of the distribution j at location i (we will assume $0 \leq a_{i,j} \leq M$ for discussions here).

Note that there is no relation between the order of arrival and the parameter i , which indexes the domain, or j , which indexes the signals. For convenience of notation, we use the index j to indicate that $a_{i,j}$ is from signal j , or is the j th tuple with index i . However, j is not generally made explicit in the stream and we assume it is not available for the processing algorithm to use. We use n_i to denote the number of tuples for index i seen so far, and $n = \sum_i n_i$ is the total number of tuples seen in the data stream. There are three parameters of the algorithm that are of interest to us: the amount of space used; the time used to process each item that arrives; and the time taken to produce the approximation of the quantity of interest. For an algorithm that works in the data stream to be of interest, the working space and per item processing time must both be sublinear in n and M , and ideally poly-logarithmic in these quantities.

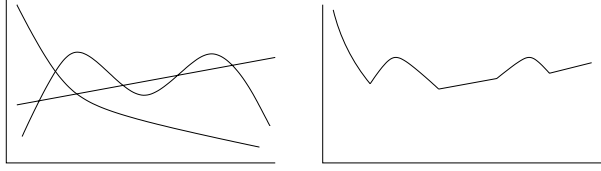


Fig. 1. A mixture of distributions (left), and their upper envelope (right)

1.2 Dominance Norms and Their Relevance

We study norms that are cumulative over the multiple distributions in the data stream. We particularly focus on the *max-dominance* defined as $\sum_i \max_j \{a_{i,j}\}$. Intuitively, this corresponds to computing the L_1 norm of the upper envelope of the distributions, illustrated schematically in Figure 1. Computing the max-dominance norm of multiple data distributions is interesting for many important reasons described below. First, applications abound where this measure is suitable for estimating the “worst case influence” under multiple distributions. For example, in the IP network scenario, i ’s correspond to source IP addresses and $a_{i,j}$ corresponds to the number of packets sent by IP address i in the j th transmission. Here the max-dominance measures the maximum possible utilization of the network if the transmissions from different source IP addresses were coordinated. A similar analysis of network capacity using max-dominance is relevant in the electrical grid [9] and in other instances with IP networks, such as using SNMP [18]. The concept of max-dominance occurs in financial applications, where the maximum dollar index (MDI) for securities class action filings characterize the intensity of litigation activity through time [21]. In addition to finding specific applications, max-dominance norm has intrinsic conceptual interest in the following two ways. (1) If $a_{i,j}$ ’s were all 0 or 1, then this norm reduces to calculating the size of the union of the multiple sets. Therefore max-dominance norm is a generalization of the standard union operation. (2) Max-dominance can be viewed as a generalization of the problem of counting the number of distinct elements i that occur within a stream. The two norms again coincide when each $a_{i,j}$ takes on binary values.

We denote the *max-dominance* of such a stream a as $\text{dom}_{\max}(a) = \sum_i \max_{1 \leq l \leq n_i} \{a_{i,l}\}$. Equivalently, we define the i ’th entry of an implicit state vector as $\max_{1 \leq l \leq n_i} \{a_{i,l}\}$, and the $\text{dom}_{\max}$ function is the L_1 norm of this vector. Closely related to max-dominance norms is *min-dominance*: $\sum_i \min_j \{a_{i,j}\}$ and *median-dominance*: $\sum_i \text{median}_j \{|a_{i,j}|\}$; or more generally $\sum_i \text{quantiles}_j \{|a_{i,j}|\}$. Generalizing these measures on various orderings (not just quantiles) of values are relative measures of dominance: *Relative count dominance* is based on counting the number of places where one distribution dominates another (or others, more generally), $|\{i | a_i > b_i\}|$ for two given data distributions a and b , and *relative sum dominance* which is $\sum_i \{\frac{a_i}{\max\{1, b_i\}}\}$. All of these dominances are very natural for collating information from two or more signals in the data stream.

1.3 Our Results

Our contributions are as follows.

1. We initiate the study of dominance norms as indicators for collating information from multiple signals in data streams.
2. We present streaming algorithms for maintaining the max-dominance of multiple data streams. Our algorithms estimate max-dominance to $1 + \epsilon$ approximation with probability at least $1 - \delta$. We show an algorithm that uses $O(\frac{\log M \log \log^2 M}{\epsilon^3})$ by reducing the problem to multiple instances of the problem of estimating distinct items on data streams. However, the main part of our technical contribution is an improved algorithm that uses only $O(\frac{\log M}{\epsilon^2})$ space. In both cases, the running time as well as time to compute the norm is also similarly polylogarithmic. This is the bulk of our technical work. No such sublinear space and time result was known for estimating any dominance norms in the data stream model.
3. We show that, in contrast, all other closely related dominance norms — min-dominance, relative count dominance and relative sum dominance — need linear space to be even probabilistically approximated in the streaming model. The full results are given in [6] and use reductions from other problems known to be hard in the streaming and communication complexity models.

1.4 Related Work

Surprisingly, almost no data stream algorithms are known for estimating any of the dominance norms, although recent work has begun to investigate the problems involved in analyzing and comparing multiple data streams [23]. There, the problem is to predict missing values, or determine variations from expected values, in multiple evolving streams. Much of the recent flurry of results in data streams has focused on using various L_p norms for *individual* distributions to compare and collate information from different data streams, for example, $(\sum_i (\sum_j a_{i,j})^p)^{1/p}$ for $0 < p \leq 2$ [1,11,17] and related notions such as Hamming norms $\sum_i (\sum_j a_{i,j} \neq 0)$ [4]. While these norms are suitable for capturing comparative trends in multiple data streams, they are not applicable for computing the various dominance norms (max, min, count or relative). Most related to the methods here is our work in [4], where we used Stable Distributions with small parameter. We extend this work by applying it to a new scenario, that of dominance norms. Here we need to derive new properties: the behavior of these distributions as the parameter approaches zero (Sections 3.4—3.6), how range sums of variables can be computed efficiently (Section 3.6), and so on.

Also relevant is work on computing the number of distinct values within a stream, which has been the subject of much study [13,14,15,4,2]. In [15], the authors mention that their algorithm can be applied to the problem of computing $\sum_i \max\{a_i, b_i\}$, which is a restricted version of our notion of dominance norm. Applying their algorithm directly yields a time cost of $\Omega(M)$ to process each item, which is prohibitive for large M (it is exponential in the input size). Other approaches which are used in ensemble as indicators when observing data streams include monitoring those items that occur very frequently (the “heavy hitters” of [12,10]), and those that occur very rarely [8]. We mention only

work related to our current interest in computing dominance norms of streams. For a more general overview of issues and algorithms in processing data streams, see the survey [19].

2 Max-Dominance Norms Using Distinct Element Counting

Let us first elaborate on the challenge in computing dominance norms of multiple data streams by focusing on the max-dominance norm. If we had space proportional to the range of values i then for each i , we can store $\max_j \{|a_{i,j}|\}$ for all the $a_{i,j}$'s seen thus far, and incrementally maintain $\sum_i \max_j \{|a_{i,j}|\}$. However, in our motivating scenarios, algorithms for computing max-dominance norms are no longer obvious.

Theorem 1. *By maintaining $\frac{\log M}{\epsilon}$ independent copies of a method for counting distinct values in the stream, we can compute a $1 \pm \epsilon$ approximation to the dominance norm with probability $1 - \delta$. The per-element processing time is that needed to insert an element into $O(\log \frac{a_{i,j}}{\epsilon})$ of the distinct elements algorithms.*

Proof. We require access to $K = \lceil \log(M)/\log(1 + \epsilon) \rceil + 1 = \log(\frac{M}{\epsilon}) + O(1)$ different instantiations of distinct elements algorithms. We shall refer to these as $D_0 \dots D_k \dots D_K$. On receiving a tuple $(i, a_{i,j})$, we compute the 'level', l , of this item as $l = \lceil \frac{\log a_{i,j}}{\log(1+\epsilon)} \rceil$. We then insert the identifier i into certain of the distinct element algorithms: those D_k where $0 \leq k \leq l$. Let D_k^{out} indicate the approximation of the number of distinct elements of D_k . The approximation of the dominance norm of the sequence is given by:

$$\hat{d}(a) = D_0^{out} + \sum_{j=1}^K (\lceil (1 + \epsilon)^j \rceil - \lceil (1 + \epsilon)^{j-1} \rceil) D_j^{out}$$

We consider the effect of any individual i , which is represented in the stream by multiple values $a_{i,j}$. By the effect of the distinct elements algorithms, the contribution is 1 at each level up to $\lceil \log(\max_j \{a_{i,j}\}) / \log(1 + \epsilon) \rceil$. The effect of this on the scaled sum is then between $\max_j \{a_{i,j}\}$ and $(1 + \epsilon) \max_j \{a_{i,j}\}$ if each distinct element algorithms give the exact answer. This procedure is illustrated graphically in Figure 2. Since these are actually approximate, then we find a result between $(1 - \epsilon) \max_j \{a_{i,j}\}$ and $(1 + \epsilon)^2 \max_j \{a_{i,j}\}$. Summing this over all i , we get $(1 - \epsilon) \text{dom}_{\max}(a) \leq \hat{d}(a) \leq (1 + \epsilon)^2 \text{dom}_{\max}(a)$

Corollary 1. *There is an algorithm for computing Dominance norms which outputs a $(1 + \epsilon)$ approximation with probability $1 - \delta$ which uses $\tilde{O}(\frac{\log M}{\epsilon} (\frac{1}{\epsilon^2} + \log M) \log \frac{1}{\delta})$ space, and amortized time $\tilde{O}(\frac{1}{\epsilon} \log^2 M \log \frac{\log M}{\delta \epsilon})$ per item (here $\tilde{O}$ suppresses $\log \log n$ and $\log \frac{1}{\epsilon}$ factors).*

This follows by adopting the third method described in [2], which is the most space efficient method for finding the number of distinct elements in a stream that is in the literature. The space required for each D is $O((\frac{1}{\epsilon^2} + \log M) \log \frac{1}{\epsilon} \log \log(M) \log \frac{1}{\delta})$. Updates take amortized time $O(\log M + \log \frac{1}{\epsilon})$. In order to have probability $1 - \delta$ of

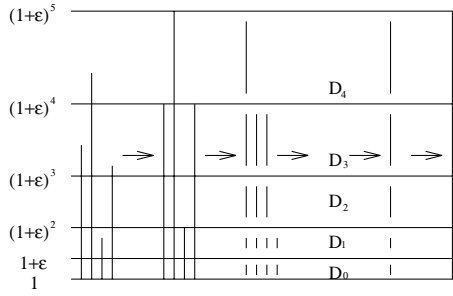


Fig. 2. Each item is rounded to the next value of $(1+\epsilon)^l$. We send every $k < l$ to a distinct elements counter D_k . The output of these counters is scaled appropriately, to get back the maximum value seen (approximate to $1 + \epsilon$)

every count being accurate within the desired bounds, we have to increase the accuracy of each individual test, replacing δ with $\frac{\epsilon\delta}{\log M}$. Putting this together with the above theorem gets the desired result. Clearly, with better methods for computing the number of distinct elements, better results could be obtained. $\square$

3 Max-Dominance Norm via Stable Distributions

We present a second method to compute the max-dominance of streams, which makes use of *stable distributions* to improve the space requirements.

3.1 Stable Distributions

Indyk pioneered the use of Stable Distributions in data streams and since then have received a great deal of attention [17,5,4,16]. Throughout our discussion of distributions, we shall use $\sim$ for the equivalence relation meaning “is equivalent in distribution to”.

A stable distribution is defined by four parameters. These are (i) the stability index, $0 < \alpha \leq 2$; (ii) the skewness parameter, $-1 \leq \beta \leq 1$; (iii) scale parameter, $\gamma > 0$; and (iv) location parameter, δ . Throughout we shall deal with a canonical representation of stable distributions, where $\gamma = 1$ and $\delta = 0$. Therefore, we consider stable distributions $S(\alpha, \beta)$ so that, given α and β the distribution is uniquely defined by these parameters. We write $X \sim S(\alpha, \beta)$ to denote the random variable X is distributed as a stable distribution with parameters α and β . When $\beta = 0$, as we will often find, then the distribution is symmetric about the mean, and is called *strictly stable*.

Definition 1. The strictly stable distribution $S(\alpha, 0)$ is defined by the property that given independent variables distributed stable:

$$X \sim S(\alpha, 0), Y \sim S(\alpha, 0), Z \sim S(\alpha, 0) \Rightarrow aX + bY \sim cZ, a^\alpha + b^\alpha = c^\alpha$$

That is, if X and Y are distributed with stability parameter α , then any linear combination of them is also distributed as a stable distribution with the same stability parameter α .

The result is scaled by the scalar c where $c = |a^\alpha + b^\alpha|^{1/\alpha}$. The definition uniquely defines a distribution, up to scaling and shifting. By centering the distribution on zero and fixing a scale, we can talk about *the* strictly stable distribution with index α . From the definition it follows that (writing $\|\mathbf{a}\|_\alpha = (\sum_i |a_i|^\alpha)^{1/\alpha}$)

$$X_1 \dots X_n \sim S(\alpha, 0); \mathbf{a} = (a_1, \dots, a_n); \Rightarrow \sum_i a_i X_i \sim \|\mathbf{a}\|_\alpha S(\alpha, 0)$$

3.2 Our Result

Recall that we wish to compute the sum of the maximum values seen in the stream. That is, we want to find $\text{dom}_{\max}(a) = \sum_i \max\{a_{i,1}, a_{i,2}, \dots, a_{i,n_i}\}$. We will show how the max-dominance can be found approximately by using values drawn from stable distributions. This allows us to state our main theorem:

Theorem 2. *It is possible to compute an approximation to $\sum_i \max_{1 \leq j \leq n_i} \{a_{i,j}\}$ in the streaming model that is correct within a factor of $(1 + \epsilon)$ with probability $1 - \delta$ using space $O(\frac{1}{\epsilon^2} (\log(M) + \epsilon^{-1} \log n \log \log n) \log \frac{1}{\delta})$ and taking $O(\frac{1}{\epsilon^4} \log a_{i,j} \log n \log \frac{1}{\delta})$ time per item.*

3.3 Idealized Algorithm

We first give an outline algorithm, then go on to show how this algorithm can be applied in practice on the stream with small memory and time requirements. We imagine that we have access to a special indicator distribution X . This has the (impossible) property that for any positive integer c (that is, $c > 0$) then $E(cX) = 1$ and bounded variance. From this it is possible to derive a solution problem of finding the max-dominance of a stream of values. We maintain a scalar z , initially zero. We create a set of $x_{i,k}$, each drawn from iid distributions $X_{i,k} \sim X$. For every $a_{i,j}$ in the input streams we update z as follows:

$$z \leftarrow z + \sum_{k=1}^{a_{i,j}} x_{i,k}$$

This maintains the property that the expectation of z is $\sum_i \max_j \{a_{i,j}\}$, as required. This is a consequence of the “impossible” property of $X_{i,k}$ that it contributes only 1 to the expectation of z no matter how many times it is added. For example, suppose our stream consists of $\{(i = 1, a_{1,1} = 2), (3, 3), (3, 5)\}$. Then z is distributed as $X_{1,1} + X_{1,2} + 2X_{3,1} + 2X_{3,2} + 2X_{3,3} + X_{3,4} + X_{3,5}$. The expected value of z is then the number of different terms, 7, which is the max dominance that we require (2+5). The required accuracy can be achieved by in parallel keeping several different values of z based on independent drawings of values for $x_{i,k}$. There are a number of hurdles to overcome in order to turn this idea into a practical solution.

1. **How to choose the distributions $X_{i,k}$?** We shall see how appropriate use of stable distributions can achieve a good approximation to these indicator variables.
2. **How to reduce space requirements?** The above algorithm requires repeated access to $x_{i,k}$ for many values of i and k . We need to be able to provide this access without explicitly storing every $x_{i,k}$ that is used. We also need to show that the required accuracy can be achieved by carrying out only a small number of independent repetitions in parallel.

3. **How to compute efficiently?** We require fast per item processing, that is polylogarithmic in the size of the stream and the size of the items in the stream. But the algorithm above requires adding $a_{i,j}$ different values to a counter in each step: time linear in the size of the data item (that is, exponential in the size of its binary representation). We show how to compute the necessary range sums efficiently while ensuring that the memory usage remains limited.

3.4 Our Algorithm

We will use stable distributions with small stability parameter α in order to *approximate* the indicator variable $X_{i,k}$. Stable distributions can be used to approximate the number of non-zero values in a vector [4]. For each index i , we can consider a vector $\mathbf{a}(i)$ defined by the tuples for that index i along, so that $\mathbf{a}(i)_k = |\{j | a_{i,j} \geq k\}|$. Then the number of non-zero entries of $\mathbf{a}(i) = \max_j a_{i,j}$. We shall write $\mathbf{a}$ for the vector formed by concatenated all such vectors for different i . This is an alternate representation of the stream, a . To approximate the max-dominance, we will maintain a sketch vector $\mathbf{z}(a)$ which summarizes the stream a .

Definition 2. The sketch vector $\mathbf{z}(a)$ has a number of entries $m (= O(\frac{1}{\epsilon^2} \log \frac{1}{\delta}))$. We make use of a number of values $x_{i,k,l}$, each of which is drawn independently from $S(\alpha, 0)$, for $\alpha = \epsilon / \log n$. Initially $\mathbf{z}$ is set to zero in every dimension.

Invariant. We maintain the property for each l that $\mathbf{z}_l = \sum_{i,j} \sum_{k=1}^{a_{i,j}} x_{i,k,l}$

Update Procedure. On receiving each pair $(i, a_{i,j})$ in the stream, we maintain the invariant by updating $\mathbf{z}$ as follows: $\forall 1 \leq l \leq m. \mathbf{z}_l \leftarrow \mathbf{z}_l + \sum_{k=1}^{a_{i,j}} x_{i,k,l}$

Output. Our approximation of the max-dominance norm is $\ln 2 (\text{median}_l |\mathbf{z}_l|)^\alpha$

At any point, it is possible to extract from the sketch $\mathbf{z}(a)$ a good approximation of the sum of the maximum values.

Theorem 3. In the limit, as α tends to zero,

$$(1 - \epsilon) \text{dom}_{\max}(a) \leq \ln 2 (\text{median}_l |\mathbf{z}(a)_l|)^\alpha \leq (1 + \epsilon)^2 \text{dom}_{\max}(a)$$

Proof. From the defining property of stable distributions (Definition 1), we know by construction that each entry of $\mathbf{z}$ is drawn from the distribution $\|\mathbf{a}\|_\alpha S(\alpha, 0)$. We know that we will add any $x_{i,k,l}$ to $\mathbf{z}_l$ at most once for each tuple in the stream, so we have an upper bound $U = n$ on each entry of $\mathbf{a}$. A simple observation is that for small enough α and an integer valued vector then the norm $\|\mathbf{a}\|_\alpha^\alpha$ (L_α norm raised to the power α , which is just $\sum_i \mathbf{a}_i^\alpha$) approximates the number of non-zero entries in the vector. Formally, if we set an upper bound U so that $\forall i. |\mathbf{a}_i| \leq U$ and fix $0 < \alpha \leq \epsilon / \log_2 U$ then

$$\begin{aligned} |\{i | \mathbf{a}_i \neq 0\}| &= \sum_{\mathbf{a}_i \neq 0} 1^\alpha \leq \sum_{\mathbf{a}_i \neq 0} |\mathbf{a}_i|^\alpha = \|\mathbf{a}\|_\alpha^\alpha \\ &\leq \sum_{\mathbf{a}_i \neq 0} U^\alpha \leq \exp(\epsilon \ln 2) |\{i | \mathbf{a}_i \neq 0\}| \\ &\leq (1 + \epsilon) |\{i | \mathbf{a}_i \neq 0\}| \end{aligned}$$

Using this, we choose α to be $\epsilon / \log_2 n$ since each value i appears at most n times within the stream, so $U = n$. This guarantees

$$\text{dom}_{\max}(a) \leq \|a\|_{\alpha}^{\alpha} \leq (1 + \epsilon) \text{dom}_{\max}(a)$$

Lemma 1. *If $X \sim S(\alpha, \beta)$ then*

$$\lim_{\alpha \rightarrow 0^+} \text{median}(|cX|^{\alpha}) = |c|^{\alpha} \text{median}(|X|^{\alpha}) = \frac{|c|^{\alpha}}{\ln 2}$$

Proof: Let E be distributed with the exponential distribution with mean one. Then $\lim_{\alpha \rightarrow 0^+} |S(\alpha, \beta)|^{\alpha} = E^{-1}$ [7]. The density of E^{-1} is $f(x) = x^{-2} \exp(-1/x)$, $x > 0$ and the cumulative density is

$$F(x) = \int_0^x f(x)dx = \exp(-1/x)$$

so in the limit, $\text{median}(E^{-1}) = F^{-1}(1/2) = 1/\ln 2$ $\square$

Consequently $\forall k. |z_k|^{\alpha} \sim \|a\|_{\alpha}^{\alpha} |X|^{\alpha}$ and $\text{median} \|a\|_{\alpha} |X|^{\alpha} \rightarrow \|a\|_{\alpha}^{\alpha} / \ln 2$. We next make use of a standard sampling result:

Lemma 2. *Let X be a distribution with cumulative density function $F(x)$. If derivative of the inverse of $F(X)$ is bounded by a constant around the median then the median of $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ samples from X is within a factor of $1 \pm \epsilon$ of $\text{median}(X)$ with probability $1 - \delta$.*

The derivative of the inverse density is indeed bounded at the median in the limit, since $F^{-1}(r) = -1/\ln r$, and $(F^{-1})'(\frac{1}{2}) < 5$. Hence for a large enough constant c , by taking a vector z with $m = \frac{c}{\epsilon^2} \log \frac{1}{\delta}$ entries, each based on an independent repetition of the above procedure, then we can approximate the desired quantity and so $(1 - \epsilon)\|a\|_{\alpha}^{\alpha} \leq (\ln 2) \text{median}_k |z_k|^{\alpha} \leq (1 + \epsilon)\|a\|_{\alpha}^{\alpha}$ with probability $1 - \delta$ by this Lemma.

Thus to find our approximation of the sum of the maximum values, we maintain the vector z as the dot product of the underlying vector a with the values drawn from stable distributions, $x_{i,k,l}$. When we take the absolute value of each entry of z and find their median, the result raised to the power α and scaled by the factor of $\ln 2$ is the approximation of $\text{dom}_{\max}(a)$.

3.5 Space Requirement

For the algorithm to be applicable in the streaming model, we need to ensure that the space requirements are minimal, and certainly sublinear in the size of the stream. Therefore, we cannot explicitly keep all the values we draw from stable distributions, yet we require the values to be the same each time the same entry is requested at different points in the algorithm. This problem can be solved by using pseudo-random generators: we do not store any $x_{i,k,l}$ explicitly, instead we create it as a pseudo-random function of k , i , l and a small number of stored random bits whenever it is needed. We need a different set of random bits in order to generate each of the m instantiations of the procedure. We therefore need only consider the space required to store the random bits, and to hold the vector z . It is known that although there is no closed form for stable distributions for general α , it is possible to draw values from such distributions for arbitrary α by using a transform from two independent uniform random variables.

Lemma 3 (Equation (2.3) of [3]). *Let U be a uniform random variable on $[0, 1]$ and Θ uniform on $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Then*

$$S(\alpha, 0) \sim \frac{\sin \alpha \Theta}{(\cos \Theta)^{1/\alpha}} \left(\frac{\cos(1 - \alpha)\Theta}{-\ln U} \right)^{\frac{1-\alpha}{\alpha}}$$

We also make use of two other results on random variables from the literature (see for example [22]), with which we will prove the space requirements for the algorithm.

Lemma 4. (i) $Y \sim S(\alpha, 1), Z \sim S(\alpha, 1) \Rightarrow 2^{-1/\alpha}(Y - Z) \sim S(\alpha, 0)$
(ii) *In the limit, the density function $f(x)$ obeys $X \sim S(\alpha, 1), \alpha \rightarrow 0^+ \Rightarrow f(x) = O(\alpha \exp(-x^{-\alpha})x^{-\alpha-1}), x > 0$*

Lemma 5. *The space requirement of this algorithm is $O(\frac{1}{\epsilon^2}(\log M + \frac{1}{\epsilon} \log n) \log \frac{1}{\delta})$ bits.*

Proof. For each repetition of the procedure, we require $O(\log n)$ random bits to instantiate the pseudo-random generators, as per [20,17]. We also need to consider the space used to represent each entry of $\mathbf{z}$. We analyze the process at each step of the algorithm: a value x is drawn (pseudo-randomly) from $S(\alpha, 0)$, and added to an entry in $\mathbf{z}$. The number of bits needed to represent this quantity is $\log_2 |x|$. Since the cumulative distribution of the limit from Lemma 4 (ii) is

$$F_{\beta=1}(x) = \int_0^x \alpha \exp(-x^{-\alpha}) x^{-\alpha-1} dx = \exp(-x^{-\alpha})$$

then $F_{\beta=1}^{-1}(r) = (\ln r^{-1})^{-1/\alpha} \quad 0 \leq r \leq 1$

So $|x| = O(F_{\beta=0}^{-1}(r)) = O(2^{-1/\alpha}(\ln r^{-1})^{-1/\alpha})$ by Lemma 4 (i). Therefore $\log_2 |x| = O(\frac{1}{\alpha} \log \ln r)$. The dependence on α is $O(\alpha^{-1})$, which was set in Theorem 3 as $\alpha \leq \epsilon / \log n$. The value of r requires a poly-log number of bits to represent, so representing x requires $O(\frac{1}{\epsilon} \log n \log \log n)$ bits. Each entry of $\mathbf{z}$ is formed by summing many such variables. The total number of summations is bounded by Mn . So the total space to represent each entry of $\mathbf{z}$ is

$$\log z_k = \tilde{O}(\log Mnx) = \tilde{O}(\log M + \log n + \frac{1}{\epsilon} \log n)$$

The total space required for all $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ entries of $\mathbf{z}$ is $O(\frac{1}{\epsilon^3} \log \frac{1}{\delta} \log n \log \log n)$ if we assume M is bounded by a polynomial in n .

3.6 Per Item Processing Time

For each item, we must compute several sums of variables drawn from stable distributions. Directly doing this will take time proportional to $a_{i,j}$. We could precompute sums of the necessary variables, but we wish to avoid explicitly storing any values of variables to ensure that the space requirement remains sublinear. However, the defining property of stable distributions is that the sum of any number of variables is distributed as a stable distribution.

Lemma 6. *The sum $\sum_{k=1}^{a_{i,j}} x_{i,k}$ can be approximated up to a factor of $1 + \epsilon$ in $O(\frac{1}{\epsilon} \log a_{i,j})$ steps.*

Proof. To give a $1 + \epsilon$ approximation imagine rounding each $a_{i,j}$ to the closest value of $\lfloor (1 + \epsilon)^s \rfloor$, guaranteeing an answer that is no more than $(1 + \epsilon)$ the true value. So we compute sums of the form $\left(\sum_{k=\lfloor (1+\epsilon)^s \rfloor + 1}^{\lfloor (1+\epsilon)^{s+1} \rfloor} x_{i,k} \right)$ which is distributed as

$$(\lfloor (1 + \epsilon)^{s+1} \rfloor - \lfloor (1 + \epsilon)^s \rfloor)^{1/\alpha} S(\alpha, 0)$$

The sum can be computed in $\log_{1+\epsilon} a_{i,j} = \frac{\log a_{i,j}}{\log 1+\epsilon} = O(\frac{1}{\epsilon} \log a_{i,j})$ steps.

The main Theorem 2 follows as a consequence of combining Theorem 3 with Lemmas 5 and 6 and by appropriate rescaling of δ .

We briefly mention an additional property of this method, which does not follow for the previous method. It is possible to include deletions of values from the past in the following sense: if we are presented with a tuple $(i, -a_{i,j})$, then we interpret this as a request to remove the contribution of $a_{i,j}$ from index i . Provided that there was an earlier tuple $(i, a_{i,j})$, then we can compute the effect this had on z and remove this by subtracting the appropriate quantities.

4 Hardness of Other Dominances

We recall the definition of the min-dominance, $\sum_i \min_j \{a_{i,j}\}$. We show that, unlike the max-dominance norm, it is not possible to compute a useful approximation to this quantity in the data stream model. This is shown by using a reduction from the size of the intersection of two sets, a problem that is known to be hard to approximate in the communication complexity model. Similarly, finding the accumulation of any averaging function (Mean, Median or Mode) of a mixture of signals requires as much storage as there are different signals. Proofs are omitted for space reasons, see [6] for full details.

- Any algorithm to compute a constant factor approximation to min-dominance of a stream with constant probability requires $\Omega(n)$ bits of storage.
- Computing $\sum_i (\sum_j a_{i,j} / n_i)$ on the stream to any constant factor c with constant probability requires $\Omega(n/c)$ bits of storage.
- Computing $\sum_i \text{median}_j \{a_{i,j}\}$ and $\sum_i \text{mode}_j \{a_{i,j}\}$ to any constant factor with constant probability requires $\Omega(n)$ bits of memory.
- Approximating the relative sum dominance $\sum a_i / \max\{1, b_i\}$ to any constant c with constant probability requires $\Omega(n/c)$ bits of storage.

5 Conclusion

Data streams often consist of multiple signals. We initiated the study of estimating dominance norms over multiple signals. We presented algorithms for estimating the max-dominance of the multiple signals that uses small (poly-logarithmic) space and takes small time per operation. These are the first known algorithm for any dominance

norm in the data stream model. In contrast, we showed that related quantities such as the min-dominance cannot be so approximated.

We have already discussed some of the applications of max-dominance, and we expect it to find many other uses, as such, and variations thereof. The question of finding useful indicators for actionable events based on multiple data streams is an important one, and it is of interest to determine other measures which can be computed efficiently to give meaningful indicators. The analysis that we give to demonstrate the behavior of stable distributions with small index parameter α , and our procedure for summing large ranges of such variables very quickly may spur the discovery of further applications of these remarkable distributions.

Acknowledgments. We thank Mayur Datar and Piotr Indyk for some helpful discussions.

References

1. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996. Journal version appeared in *JCSS: Journal of Computer and System Sciences*, 58:137–147, 1999.
2. Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisian. Counting distinct elements in a data stream. In *Proceedings of RANDOM 2002*, pages 1–10, 2002.
3. J.M. Chambers, C.L. Mallows, and B.W. Stuck. A method for simulating stable random variables. *Journal of the American Statistical Association*, 71(354):340–344, 1976.
4. G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan. Comparing data streams using Hamming norms. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 335–345, 2002. Journal version appeared in *IEEE Transactions on Knowledge and Data Engineering*, 2003.
5. G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of tabular data via approximate distance computations. In *Proceedings of the International Conference on Data Engineering*, pages 605–616, 2002.
6. G. Cormode and S. Muthukrishnan. Estimating dominance norms of multiple data streams. Technical Report 2002-35, DIMACS, 2002.
7. N. Cressie. A note on the behaviour of the stable distributions for small index α . *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 33:61–64, 1975.
8. M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proceedings of 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 323–334, 2002.
9. <http://energycrisis.lbl.gov/>.
10. C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the First ACM SIGCOMM Internet Measurement Workshop (IMW-01)*, pages 75–82, 2001.
11. J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L_1 -difference algorithm for massive data streams. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, pages 501–511, 1999.
12. A. Feldmann, A. G. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: Methodology and experience. In *Proceedings of SIGCOMM*, pages 257–270, 2000.

13. P. Flajolet and G. N. Martin. Probabilistic counting. In *24th Annual Symposium on Foundations of Computer Science*, pages 76–82, 1983. Journal version appeared in *Journal of Computer and System Sciences*, 31:182–209, 1985.
14. P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *27th International Conference on Very Large Databases*, pages 541–550, 2001.
15. P. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architectures*, pages 281–290, 2001.
16. A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 389–398, 2002.
17. P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, pages 189–197, 2000.
18. Large-scale communication networks: Topology, routing, traffic, and control.
http://ipam.ucla.edu/programs/cntop/cntop_schedule.html.
19. S. Muthukrishnan. Data streams: Algorithms and applications. In *ACM-SIAM Symposium on Discrete Algorithms*, <http://athos.rutgers.edu/~muthu/stream-1-1.ps>, 2003.
20. N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12:449–461, 1992.
21. http://securities.stanford.edu/litigation_activity.html.
22. V. V. Uchaikin and V. M. Zolotarev. *Chance and Stability: Stable Distributions and their applications*. VSP, 1999.
23. B.-K. Yi, N. Sidiropoulos, T. Johnson, H. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. In *16th International Conference on Data Engineering (ICDE'00)*, pages 13–22, 2000.

Smoothed Motion Complexity^{*}

Valentina Damerow¹, Friedhelm Meyer auf der Heide², Harald Räcke²,
Christian Scheideler³, and Christian Sohler²

¹ PaSCo Graduate School and

² Heinz Nixdorf Institute, Paderborn University, D-33102 Paderborn, Germany
{vio, fmadh, harry, csohler}@upb.de

³ Dept. of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA,
scheideler@cs.jhu.edu

Abstract. We propose a new complexity measure for movement of objects, the *smoothed motion complexity*. Many applications are based on algorithms dealing with moving objects, but usually data of moving objects is inherently noisy due to measurement errors. Smoothed motion complexity considers this imprecise information and uses *smoothed analysis* [13] to model noisy data. The input is object to slight random perturbation and the *smoothed complexity* is the worst case expected complexity over all inputs w.r.t. the random noise. We think that the usually applied worst case analysis of algorithms dealing with moving objects, e.g., kinetic data structures, often does not reflect the real world behavior and that smoothed motion complexity is much better suited to estimate dynamics.

We illustrate this approach on the problem of maintaining an orthogonal bounding box of a set of n points in $\mathbb{R}^d$ under linear motion. We assume speed vectors and initial positions from $[-1, 1]^d$. The motion complexity is then the number of combinatorial changes to the description of the bounding box. Under perturbation with Gaussian normal noise of deviation σ the smoothed motion complexity is only polylogarithmic: $O(d \cdot (1 + 1/\sigma) \cdot \log n^{3/2})$ and $\Omega(d \cdot \sqrt{\log n})$. We also consider the case when only very little information about the noise distribution is known. We assume that the density function is monotonically increasing on $\mathbb{R}_{\leq 0}$ and monotonically decreasing on $\mathbb{R}_{\geq 0}$ and bounded by some value C . Then the motion complexity is $O(\sqrt{n \log n} \cdot C + \log n)$ and $\Omega(d \cdot \min\{\sqrt[5]{n}/\sigma, n\})$.

Keywords: Randomization, Kinetic Data Structures, Smoothed Analysis

1 Introduction

The task to process a set of continuously moving objects arises in a broad variety of applications, e.g., in mobile ad-hoc networks, traffic control systems, and computer graphics (rendering moving objects). Therefore, researchers investigated data structures that can be efficiently maintained under continuous motion, e.g., to answer proximity queries [5], maintain a clustering [8], a convex hull [4], or some connectivity information of the moving point set [9]. Within the framework of kinetic data structures the efficiency of

^{*} The third and the fifth author are partially supported by DFG-Sonderforschungsbereich 376, DFG grant 872/8-1, and the Future and Emerging Technologies program of the EU under contract number IST-1999-14186 (ALCOM-FT).

such a data structure is analyzed w.r.t. to the worst case number of combinatorial changes in the description of the maintained structure that occur during linear (or low degree algebraic) motion. These changes are called (*external*) *events*. For example, to maintain the smallest orthogonal bounding box of a point set in $\mathbb{R}^d$ has a unique description at a certain point of time consisting of the $2d$ points that attain the minimum and maximum value in each of the d coordinates. If any such minimum/maximum point changes then an event occurs. We call the worst case number of events w.r.t. the maintenance of a certain structure under linear motion the *worst case motion complexity*.

We introduce an alternative measure for the dynamics of moving data called the *smoothed motion complexity*. Our measure is based on *smoothed analysis*, a hybrid between worst case analysis and average case analysis. Smoothed analysis has been introduced by Spielman and Teng [13] in order to explain the typically good performance of the simplex algorithm on almost every input. It asks for the worst case expected performance over all inputs where the expectation is taken w.r.t. small random noise added to the input. In the context of mobile data this means that both the speed value and the starting position of an input configuration are slightly perturbed by random noise. Thus the smoothed motion complexity is the worst case expected motion complexity over all inputs perturbed in such a way. Smoothed motion complexity is a very natural measure for the dynamics of mobile data since in many applications the exact position of mobile data cannot be determined due to errors caused by physical measurements or fixed precision arithmetic. This is, e.g., the case when the positions of the moving objects are determined via GPS, sensors, and basically in any application involving 'real life' data.

We illustrate our approach on the problem to maintain the smallest orthogonal bounding box of a point set moving in $\mathbb{R}^d$. The bounding box is a fundamental measure for the extend of a point set and it is useful in many applications, e.g., to estimate the sample size in sublinear clustering algorithms [3], in the construction of R -trees, for collision detection, and visibility culling.

1.1 The Problem Statement

We are given a set P of n points in $\mathbb{R}^d$. The position $\text{pos}_i(t)$ of the i th point at time t is given by a linear function of t . Thus we have $\text{pos}_i(t) = s_i \cdot t + p_i$ where p_i is the initial position and s_i the speed. We normalize the speed vectors and initial positions such that $p_i, s_i \in [-1, 1]^d$.

The motion complexity of the problem is the number of combinatorial changes to the set of $2d$ extreme points defining the bounding box. Clearly this motion complexity is $O(d \cdot n)$ in the worst case, 0 in the best case, and $O(d \cdot \log n)$ in the average case. When we consider smoothed motion complexity we add to each coordinate of the speed vector and each coordinate of the initial position an i.i.d. random variable from a certain probability distribution, e.g., Gaussian normal distribution. Then the smoothed motion complexity is the worst case expected complexity over all choices of p_i and s_i .

1.2 Related Work

In [4] Basch et al. introduced *kinetic data structures* (KDS) which is a framework for data structures for moving objects. In KDS the (near) future motion of all objects is known and can be specified by so-called pseudo-algebraic functions of time specified by linear functions or low-degree polynomials. This specification is called a *flight plan*. The goal is to maintain the description of a combinatorial structure as the objects move according to this flight plan. The flight plan may change from time to time and these updates are reported to the KDS. The efficiency of a KDS is analyzed by comparing the worst case number of internal (events needed to maintain auxiliary data structures) and external events it processed against the worst case number of external events. Using this framework many interesting kinetic data structures have been developed, e.g., for connectivity of discs [7] and rectangles [9], convex hulls [4], proximity problems [5], and collision detection for simple polygons [10]. In [4] the authors developed a KDS to maintain a bounding box of a moving point set in $\mathbb{R}^d$. The number of events these data structures process is $O(n \log n)$ which is close to the worst case motion complexity of $\Theta(n)$. In [1] the authors showed that it is possible to maintain an $(1 + \epsilon)$ -approximation of such a bounding box. The advantage of this approach is that the motion complexity of this approximation is only $O(1/\sqrt{\epsilon})$. The average case motion complexity has also been considered in the past. If n particles are drawn independently from the unit square then it has been shown that the expected number of combinatorial changes in the convex hull is $\Theta(\log^2(n))$, in the Voronoi diagram $\Theta(n^{3/2})$ and in the closest pair $\Theta(n)$ [15].

Smoothed analysis has been introduced by Spielman and Teng [13] to explain the polynomial run time of the simplex algorithm on inputs arising in applications. They showed that the smoothed run time of the shadow-vertex simplex algorithm is polynomial in the input size and $1/\sigma$. In many follow-up papers other algorithms and values have been analyzed via smoothed analysis, e.g., the perceptron algorithm [6], condition numbers of matrices [12], quicksort, left-to-right maxima, and shortest paths [2]. Recently, smoothed analysis has been used to show that many existing property testing algorithms can be viewed as sublinear decision algorithms with low smoothed error probability [14]. In [2] the authors analyzed the smoothed number of left-to-right maxima of a sequence of n numbers. We will use the left-to-right maxima problem as an auxiliary problem but we will use a perturbation scheme that fundamentally differs from that analyzed in [2].

1.3 Our Results

Typically, measurement errors are modelled by the Gaussian normal distribution and so we analyze the smoothed complexity w.r.t. Gaussian normally distributed noise with deviation σ . We show that the smoothed motion complexity of a bounding box under Gaussian noise is $O(d \cdot (1 + 1/\sigma) \cdot \log n^{3/2})$ and $\Omega(d \cdot \sqrt{\log n})$. In order to get a more general result we consider monotone probability distributions, i.e., distributions where the density function f is bounded by some constant C and monotonically increasing on $\mathbb{R}_{\leq 0}$ and monotonically decreasing on $\mathbb{R}_{\geq 0}$. Then the smoothed motion complexity is $O(d \cdot (\sqrt{n \log n} \cdot C + \log n))$. Polynomial smoothed motion complexity is, e.g., attained by the uniform distribution where we obtain a lower bound of $\Omega(d \cdot \min\{\sqrt[5]{n}/\sigma, n\})$.

Note that in the case of speed vectors from some arbitrary range $[-S, S]^d$ instead of $[-1, 1]^d$ the above upper bounds hold if we replace σ by σ/S .

These results make it very unlikely, that in a typical application the worst case bound of $\Theta(d \cdot n)$ is attained. As a consequence, it seems reasonable to analyze KDS's w.r.t. the smoothed motion complexity rather than the worst case motion complexity.

Our upper bounds are obtained by analyzing a related auxiliary problem: the smoothed number of left-to-right maxima in a sequence of n numbers. For this problem we also obtained lower bounds which only can be stated here: in the case of uniform noise we have $\Omega(\sqrt{n/\sigma})$ and in the case of normally distributed noise we can apply the average case bound of $\Omega(\log n)$. These bounds differ only by a factor of $\sqrt{\log n}$ from the corresponding upper bounds. In the second case the bounds are even tight for constant σ . Therefore, we can conclude that our analysis is tight w.r.t. the number of left-to-right maxima. To obtain better results a different approach that does not use left-to-right maxima as an auxiliary problem is necessary.

2 Upper Bounds

To show upper bounds for the number of external events while maintaining the bounding box for a set of moving points we make the following simplifications. We only consider the 1D problem. Since all dimensions are independently from each other an upper or lower bound for the 1D problem can be multiplied by d to yield a bound for the problem in d dimensions.

Further, we assume that the points are ordered by their increasing initial positions and that they are all moving to the left with absolute speed values between 0 and 1. We only count events that occur because the leftmost point of the 1D bounding box changes. Note that these simplifications do not asymptotically affect the results in this paper.

A necessary condition for the j th point to cause an external event is that all its *preceding* points have smaller absolute speed values, i.e. that $s_i < s_j$, $\forall i < j$. If this is the case we call s_j a left-to-right maximum. Since we are interested in an upper bound we can neglect the initial positions of the points and need only to focus on the sequence of absolute speed values $S = (s_1, \dots, s_n)$ and count the left-to-right maxima in this sequence.

The general concept for estimating the number of left-to-right maxima within the sequence is as follows. Let f and F denote the density function and distribution function, respectively, of the noise that is added to the initial speed values. (This means $\tilde{s}_i = s_i + \phi_i$ where ϕ_i is chosen according to density function f .)

Let $\Pr[\text{LTR}_j]$ denote the probability that $\tilde{s}_j$ is a left-to-right maximum. We can write this probability as

$$\Pr[\text{LTR}_j] = \int_{-\infty}^{\infty} \prod_{i=1}^{j-1} F(x - s_i) \cdot f(x - s_j) dx \quad . \quad (1)$$

This holds since $F(x - s_i)$ is the probability that the i th element is not greater than x after the perturbation. Since all perturbations are independently from each other, $\prod_{i=1}^{j-1} F(x - s_i)$ is the probability that *all* elements preceding $\tilde{s}_j$ are below x . Consequently, $\int_{-\infty}^{\infty} \prod_{i=1}^{j-1} F(x - s_i) \cdot f(x - s_j) dx$ can be interpreted as the probability that the

j th element reaches x and is a left-to-right maximum. Hence, integration over x gives the probability $\Pr[\text{LTR}_j]$.

In the following we describe how to derive a bound on the above integral. First suppose that all s_i are equal, i.e., $s_i = s$ for all i . Then $\Pr[\text{LTR}_j] = \int_{-\infty}^{\infty} F(x-s)^{j-1} \cdot f(x-s) dx = \int_0^1 z^{j-1} dz = 1/j$, where we substituted $z := F(x-s)$. (Note that this result only reveals the fact that the probability for the j th element to be the largest is $1/j$.)

Now, suppose that the speed values are not equal but come from some interval $[s_{\min}, s_{\max}]$. In this case $\Pr[\text{LTR}_j]$ can be estimated by

$$\begin{aligned} \Pr[\text{LTR}_j] &= \int_{-\infty}^{\infty} \prod_{i=1}^{j-1} F(x-s_i) \cdot f(x-s_j) dx \\ &\leq \int_{-\infty}^{\infty} F(x-s_{\min})^{j-1} \cdot f(x-s_{\max}) dx \\ &= \int_{-\infty}^{\infty} F(z+\delta)^{j-1} f(z) dz, \end{aligned}$$

where we use δ to denote $s_{\max} - s_{\min}$. Let $Z_{\delta,r}^f := \{z \in \mathbb{R} \mid f(z)/f(z+\delta) \geq r\}$ denote the subset of $\mathbb{R}$ that contains all elements z for which the ratio $f(z)/f(z+\delta)$ is larger than r . Using this notation we get

$$\begin{aligned} \Pr[\text{LTR}_j] &\leq \int_{\mathbb{R} \setminus Z_{\delta,r}^f} F(z+\delta)^{j-1} f(z) dz + \int_{Z_{\delta,r}^f} F(z+\delta)^{j-1} f(z) dz \\ &\leq \int_{\mathbb{R} \setminus Z_{\delta,r}^f} F(z+\delta)^{j-1} \frac{f(z)}{f(z+\delta)} f(z+\delta) dz + \int_{Z_{\delta,r}^f} f(z) dz \\ &\leq r \cdot \int_{\mathbb{R} \setminus Z_{\delta,r}^f} F(z+\delta)^{j-1} f(z+\delta) dz + \int_{Z_{\delta,r}^f} f(z) dz \\ &\leq r \cdot \frac{1}{j} + \int_{Z_{\delta,r}^f} f(z) dz. \end{aligned} \tag{2}$$

Now, we can formulate the following lemma.

Lemma 1. *Let f denote the density function of the noise distribution and define for positive parameters δ and r the set $Z_{\delta,r}^f \subseteq \mathbb{R}$ as $Z_{\delta,r}^f := \{z \in \mathbb{R} \mid f(z)/f(z+\delta) \geq r\}$. Further, let Z denote the probability of the set $Z_{\delta,r}^f$ with respect to f , i.e., $Z := \int_{Z_{\delta,r}^f} f(z) dz$. Then the number of left-to-right maxima in a sequence of n elements that are perturbed with noise distribution F is at most*

$$r \cdot \lceil 1/\delta \rceil \cdot \log n + n \cdot Z.$$

Proof. We are given an input sequence S of n speed values from $(0, 1]$. Let $\mathcal{L}(S)$ denote the expected number of left-to-right maxima in the corresponding sequence of speed values perturbed with noise distribution f . We are interested in an upper bound on this value. The following claim shows that we only need to consider input sequences of monotonically increasing speed values.

Claim. The maximum expected number of left-to-right maxima in a sequence of n perturbed speed values is obtained for an input sequence S of initial speed values that is monotonically increasing. $\square$

From now on we assume that S is a sequence of monotonically increasing speed values. We split S into $\lceil 1/\delta \rceil$ subsequences such that the ℓ th subsequence S_ℓ , $\ell \in \{1, \dots, \lceil 1/\delta \rceil\}$ contains all speed values between $(\ell - 1)\delta$ and $\ell\delta$, i.e., $S_\ell := (s \in S : (\ell - 1) \cdot \delta < s \leq \ell \cdot \delta)$. Note that each subsequence is monotonically increasing.

Let $\mathcal{L}(S_\ell)$ denote the expected number of left-to-right maxima in subsequence S_ℓ . Now we first derive a bound on each $\mathcal{L}(S_\ell)$ and then we utilize $\mathcal{L}(S) \leq \sum_\ell \mathcal{L}(S_\ell)$ to get an upper bound on $\mathcal{L}(S)$.

Fix $\ell \in \{1, \dots, \lceil 1/\delta \rceil\}$. Let k_ℓ denote the number of elements in subsequence S_ℓ . We have

$$\mathcal{L}(S_\ell) = \sum_{j=1}^{k_\ell} \Pr[\text{LTR}_j] ,$$

where $\Pr[\text{LTR}_j]$ is the probability that the j th element of subsequence S_ℓ is a left-to-right maximum within this subsequence. We can utilize Inequality 2 for $\Pr[\text{LTR}_j]$ because the initial speed values in a subsequence differ at most by δ . This gives

$$\mathcal{L}(S_\ell) \leq \sum_{j=1}^{k_\ell} \left(r \cdot \frac{1}{j} + Z \right) \leq r \cdot \log n + k_\ell \cdot Z .$$

Hence, $\mathcal{L}(S) \leq \sum_\ell \mathcal{L}(S_\ell) \leq r \cdot \lceil 1/\delta \rceil \cdot \log n + n \cdot Z$, as desired. $\square$

2.1 Normally Distributed Noise

In this section we show how to apply the above lemma to the case of normally distributed noise. We prove the following theorem.

Theorem 1. *The expected number of left-to-right maxima in a sequence of n speed values perturbed by random noise from the standard normal distribution $N(0, \sigma)$ is $O(\frac{1}{\sigma} \cdot (\log n)^{3/2} + \log n)$.*

Proof. Let $\varphi(z) := \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{z^2}{2\sigma^2}}$ denote the standard normal density function with expectation 0 and variance σ^2 . In order to utilize lemma 1 we choose $\delta := \frac{\sigma}{\sqrt{\log n}}$. For $z \leq 2\sigma\sqrt{\log n}$ it holds that

$$\varphi(z)/\varphi(z + \delta) = e^{(\delta/\sigma^2) \cdot z + \delta^2/(2\sigma^2)} = e^{z/(\sigma\sqrt{\log n}) + 1/(2\log n)} \leq e^3 .$$

Therefore, if we choose $r := e^3$ we have $Z_{\delta,r}^\varphi \subset [2\sigma\sqrt{\log n}, \infty)$. Now, we derive a bound on $\int_{Z_{\delta,r}^\varphi} \varphi(z) dz$. It is well known from probability theory that for the normal density function with expectation 0 and variance σ^2 it holds that $\int_{k\sigma}^\infty \varphi(z) dz \leq e^{-k^2/4}$. Hence,

$$\int_{Z_{\delta,r}^\varphi} \varphi(z) dz \leq \int_{2\sigma\sqrt{\log n}}^\infty \varphi(z) dz \leq \frac{1}{n} .$$

Altogether we can apply Lemma 1 with $\delta = \sigma/\sqrt{\log n}$, $r = e^3$ and $Z = 1/n$. This gives that the number of left-to-right maxima is at most $O(\frac{1}{\sigma} \cdot (\log n)^{3/2} + \log(n))$, as desired. $\square$

2.2 Monotonic Noise Distributions

In this section we investigate upper bounds for general noise distributions. We call a noise distribution *monotonic* if the corresponding density function is monotonically increasing on $\mathbb{R}_{\leq 0}$ and monotonically decreasing on $\mathbb{R}_{\geq 0}$. The following theorem gives an upper bound on the number of left-to-right maxima for arbitrary monotonic noise distributions.

Theorem 2. *The expected number of left-to-right maxima in a sequence of n speed values perturbed by random noise from a monotonic noise distribution is $O(\sqrt{n \log n \cdot f(0)} + \log n)$.*

Proof. Let f denote the density function of the noise distribution and let $f(0)$ denote the maximum of f . We choose $r := 2$ whereas δ will be chosen later. In order to apply Lemma 1 we only need to derive a bound on $\int_{Z_{\delta,r}^f} f(z) dz$. Therefore, we first define sets $Z_i, i \in \mathbb{N}$ such that $\cup_i Z_i \supseteq Z_{\delta,r}^f$ and then we show how to estimate $\int_{\cup_i Z_i} f(z) dz$.

First note that for $z + \delta < 0$ we have $f(z) < f(z + \delta)$ because of the monotonicity of f . Hence $Z_{\delta,r}^f \subseteq [-\delta, \infty)$. We partition $[-\delta, \infty)$ into intervals of the form $[(\ell - 1) \cdot \delta, \ell \cdot \delta]$ for $\ell \in \mathbb{N}_0$. Now, we define Z_i to be the i th interval that has a non-empty intersection with $Z_{\delta,r}^f$. (If less than i intervals have a non-empty intersection then Z_i is the empty set.) By this definition we have $\cup_i Z_i \supseteq Z_{\delta,r}^f$ as desired.

We can derive a bound on $\int_{\cup_i Z_i} f(z) dz$ as follows. Suppose that all $Z_i \subset \mathbb{R}_{\geq 0}$. Let $\hat{z}_i$ denote the start of interval Z_i . Then $\int_{Z_i} f(z) dz \leq \delta \cdot f(\hat{z}_i)$ because Z_i is an interval of length δ and the maximum density within this interval is $f(\hat{z}_i)$. Furthermore it holds that $f(\hat{z}_{i+2}) \leq \frac{1}{2} f(\hat{z}_i)$ for every $i \in \mathbb{N}$. To see this consider some $z_i \in Z_i \cap Z_{\delta,r}^f$. We have $f(\hat{z}_i) \geq f(z_i) > 2 \cdot f(z_i + \delta) \geq 2 \cdot f(\hat{z}_{i+2})$, where we utilized that $z_i \in Z_{\delta,r}^f$ and that $z_i + \delta \leq \hat{z}_{i+2}$. If $Z_1 = [-\delta, 0]$ we have $\int_{Z_1} f(z) dz \leq \delta \cdot f(0)$ for similar reasons. Now we can estimate $\int_{\cup_i Z_i} f(z) dz$ by

$$\begin{aligned} \int_{\cup_i Z_i} f(z) dz &\leq \sum_{i \in \mathbb{N}} \int_{Z_{2i-1}} f(z) dz + \sum_{i \in \mathbb{N}} \int_{Z_{2i}} f(z) dz + \int_{[-\delta, 0]} f(z) dz \\ &\leq \sum_{i \in \mathbb{N}} \frac{1}{2^{i-1}} \delta \cdot f(\hat{z}_1) + \sum_{i \in \mathbb{N}} \frac{1}{2^{i-1}} \delta \cdot f(\hat{z}_2) + \delta \cdot f(0) \\ &\leq 2\delta f(\hat{z}_1) + 2\delta f(\hat{z}_2) + \delta \cdot f(0) \leq 5\delta \cdot f(0) . \end{aligned}$$

Lemma 1 yields that the number of left-to-right maxima is at most $2 \cdot \lceil \frac{1}{\delta} \rceil \cdot \log n + n \cdot 5\delta \cdot f(0)$. Now, choosing $\delta := \sqrt{\log n / (f(0) \cdot n)}$ gives the theorem. $\square$

3 Lower Bounds

For showing lower bounds we consider the 1D problem and map each point with initial position p_i and speed s_i to a point $P_i = (p_i, s_i)$ in 2D. We utilize that the number of external events when maintaining the bounding box in 1D is strongly related to the number of vertices of the convex hull of the P_i 's. If we can arrange the points in the 2D

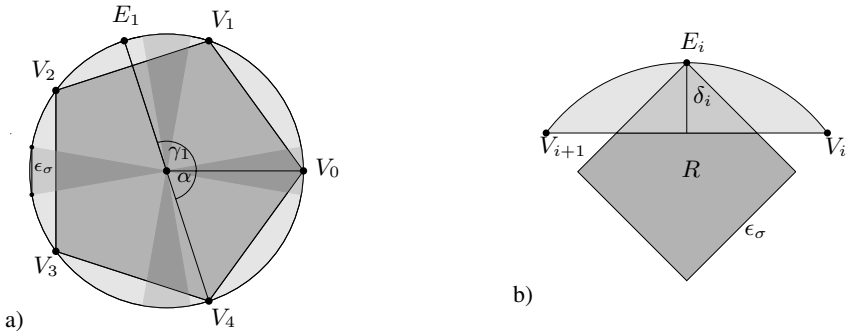


Fig. 1. (a) The partitioning of the plane into different regions. If the extreme point E_i of a boundary region i falls into the shaded area the corresponding boundary region is not valid. (b) The situation where the intersection between a boundary region i and the corresponding range square R_i is minimal.

plane such that after perturbation L points lie on the convex hull on expectation, we can deduce a lower bound of $L/2$ on the number of external events.

By this method the results of [11] directly imply a lower bound of $\Omega(\sqrt{\log n})$ for the case of normally distributed noise. For the case of monotonic noise distributions we show that the number of vertices on the convex hull is significantly larger than for the case of normally distributed noise.

We choose the uniform distribution with expectation 0 and variance σ^2 . The density function f of this distribution is

$$f(x) = \begin{cases} 1/\epsilon_\sigma & |x| \leq \epsilon_\sigma/2 \\ 0 & \text{else} \end{cases}, \text{ where } \epsilon_\sigma = \sqrt{12}\sigma.$$

We construct an input of n points that has a large expected number of vertices on the convex hull after perturbation. For this we partition the plane into different regions. We inscribe an ℓ -sided regular polygon into a unit circle centered at the origin. The interior of the polygon belongs to the *inner region* while everything outside the unit circle belongs to the *outer region*. Let $V_0, \dots, V_{\ell-1}$ denote the vertices of the polygon. The i th *boundary region* is the segment of the unit circle defined by the chord $V_i V_{i+1}$ where the indices are modulo ℓ , c.f. Figure 1a). An important property of these regions is expressed in the following observation.

Observation 1 *If no point lies in the outer region then every non-empty boundary region contains at least one point that is a vertex of the convex hull.* $\square$

In the following, we select the initial positions of the input points such that it is guaranteed that after the perturbation the outer region is empty and the expected number of non-empty boundary regions is large.

We need the following notations and definitions. For an input point j we define the range square R to be the axis-parallel square with side length ϵ_σ centered at position (p_j, s_j) . Note that for the uniform distribution with standard deviation σ the perturbed

position of j will lie in R . Further, the intersection between the circle boundary and the perpendicular bisector of the chord $V_i V_{i+1}$ is called the *extremal point* of boundary region i and is denoted with E_i . The line segment from the midpoint of the chord to E_i is denoted with δ_i , c.f. Figure 1b).

The general outline for the proof is as follows. We try for a boundary region i to place a bunch of $\frac{n}{\ell}$ input points in the plane such that a vertex of their common range square R lies in the extremal point E_i of the boundary region. Furthermore we require that no point of R lies in the outer region. If this is possible it can be shown that the range square and the boundary region have a large intersection. Therefore it will be likely that one of the $\frac{n}{\ell}$ input points corresponding to the square lies in the boundary region after perturbation. Then, we can derive a bound on the number of vertices in the convex hull by exploiting Observation 1, because we can guarantee that no perturbed point lies in the outer region.

Now, we formalize this proof. We call a boundary region i *valid* if we can place input points in the described way, i.e., such that their range square R_i is contained in the unit circle and a vertex of it lies in E_i . Then R_i is called the *range square corresponding to boundary region i* .

Lemma 2. *If $\sigma \leq 1/8$ and $\ell \geq 23$ then there are at least $\ell/2$ valid boundary regions.*

Proof. If $\sigma \leq 1/8$ then the relationship between ϵ_σ and σ gives $\epsilon_\sigma = 2\sqrt{3}\sigma \leq 1/2$. Let γ_i denote the angle of vector E_i with respect to the positive x -axis. A boundary region is valid iff $\sin(\gamma_i) \geq \epsilon_\sigma/2$ and $\cos(\gamma_i) \geq \epsilon_\sigma/2$. The invalid regions are depicted in Figure 1a). If $\epsilon_\sigma \leq 1/2$ these regions are small. To see this let β denote the central angle of each region. Then $2\sin(\beta/2) = \epsilon_\sigma \leq 1/2$ and $\beta \leq 2 \cdot \arcsin(1/4) \leq 0.51$. At most $\frac{\beta}{2\pi/\ell} + 1$ boundary regions can have their extreme point in a single invalid region. Hence the total number of invalid boundary regions is at most $4(\frac{\beta}{2\pi/\ell} + 1) \leq \ell/2$. $\square$

The next lemma shows that a valid boundary region has a large intersection with the corresponding range square.

Lemma 3. *Let R_i denote the range square corresponding to boundary region i . Then the area of the intersection between R_i and the i th boundary region is at least $\min\{(\frac{4}{\ell})^4, \epsilon_\sigma^2/2\}$ if $\ell \geq 4$.*

Proof. Let α denote the central angle of the polygon. Then $\alpha = \frac{2\pi}{\ell}$ and $\delta_i = 1 - \cos(\frac{\alpha}{2})$. By utilizing the inequality $\cos(\phi) \leq 1 - \frac{1}{2}\phi^2 + \frac{1}{24}\phi^4$ we get $\delta_i \geq \frac{11}{96}\alpha^2$ for $\alpha \leq 2$. Plugging in the value for α this gives $\delta_i \geq (\frac{4}{\ell})^2$ for $\ell \geq 4$.

The intersection between the range square and the boundary region is minimal when one diagonal of the square is parallel to δ_i , c.f. Figure 1b). Therefore, the area of the intersection is at least $\delta_i^2 \geq (\frac{4}{\ell})^4$ if $\delta_i \leq \sqrt{2}\epsilon_\sigma$ and at least $\epsilon_\sigma^2/2$ if $\delta_i \geq \sqrt{2}\epsilon_\sigma$. $\square$

Lemma 4. *If $\ell \leq \min\{\sqrt[5]{n/\epsilon_\sigma^2}, n/2\}$ then every valid boundary region is non-empty with probability at least $1 - 1/e$, after perturbation.*

Proof. We place $\frac{n}{\ell}$ input points on the center of a valid range square. The probability that none of these points lies in the boundary region after perturbation is

$$\Pr[\text{boundary region is empty}] \leq \left(1 - \frac{\min\{\delta_i^2, \epsilon_\sigma^2/2\}}{\epsilon_\sigma^2}\right)^{\frac{n}{\ell}},$$

because the area of the intersection is at least $\min\{\delta_i^2, \epsilon_\sigma^2/2\}$ and the whole area of the range square is ϵ_σ^2 . If $\delta_i^2 = \min\{\delta_i^2, \epsilon_\sigma^2/2\}$ the result follows since

$$\frac{\epsilon_\sigma^2}{\min\{\delta_i^2, \epsilon_\sigma^2/2\}} \leq \frac{\epsilon_\sigma^2}{\delta_i^2} \leq \epsilon_\sigma^2 \cdot \ell^4 = \epsilon_\sigma^2 \cdot \ell^5 / \ell \leq n / \ell.$$

Here we utilized that $\delta_i^2 \geq 1/\ell^4$ which follows from the proof of Lemma 3. In the case that $\epsilon_\sigma^2/2 = \min\{\delta_i^2, \epsilon_\sigma^2/2\}$ the result follows since $\frac{n}{\ell} \geq 2$. $\square$

Theorem 3. *If $\sigma \leq 1/8$ the smoothed worst case number of vertices on the convex hull is $\Omega(\min\{\sqrt[5]{n}/\sigma, n\})$.*

Proof. By combining Lemmas 2 and 4 with Observation 1 the theorem follows immediately if we choose $\ell = \Theta(\min\{\sqrt[5]{n}/\epsilon_\sigma, n\})$. $\square$

4 Conclusions

We introduced smoothed motion complexity as a measure for the complexity of maintaining combinatorial structures of moving data. We showed that for the problem of maintaining the bounding box of a set of points the smoothed motion complexity differs significantly from the worst case motion complexity which makes it unlikely that the worst case is attained in typical applications.

A remarkable property of our results is that they heavily depend on the probability distribution of the random noise. In particular, our upper and lower bounds show that there is an exponential gap in the number of external events between the cases of uniformly and normally distributed noise. Therefore we have identified an important sub-task when applying smoothed analysis. It is mandatory to precisely analyze the exact distribution of the random noise for a given problem since the results may vary drastically for different distributions.

References

1. AGARWAL, P., AND HAR-PELED, S. Maintaining approximate extent measures of moving points. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2001), pp. 148–157.
2. BANDERIER, C., MEHLHORN, K., AND BEIER, R. Smoothed analysis of three combinatorial problems. manuscript, 2002.
3. BAREQUET, G., AND HAR-PELED, S. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)* (1999), pp. 82–91.

4. BASCH, J., GUIBAS, L. J., AND HERSHBERGER, J. Data structures for mobile data. *Journal of Algorithms* 31, 1 (1999), 1–28.
5. BASCH, J., GUIBAS, L. J., AND ZHANG, L. Proximity problems on moving points. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry* (1997), pp. 344–351.
6. BLUM, A., AND DUNAGAN, J. Smoothed analysis of the perceptron algorithm. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2002), pp. 905–914.
7. GUIBAS, L. J., HERSHBERGER, J., SURI, S., AND ZHANG, L. Kinetic connectivity for unit disks. *Discrete & Computational Geometry* 25, 4 (2001), 591–610.
8. HAR-PELED, S. Clustering motion. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)* (2001), pp. 84–93.
9. HERSHBERGER, J., AND SURI, S. Simplified kinetic connectivity for rectangles and hypercubes. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2001), pp. 158–167.
10. KIRKPATRICK, D., SNOEYINK, J., AND SPECKMANN, B. Kinetic collision detection for simple polygons. *International Journal of Computational Geometry and Applications* 12, 1-2 (2002), 3–27.
11. RÉNYI, A., AND SULANKE, R. Über die konvexe Hülle von n zufällig gewählten Punkten. *Zentralblatt für Wahrscheinlichkeitstheorie und verwandte Gebiete* 2 (1963), 75–84.
12. SANKAR, A., SPIELMAN, D., AND S.TENG. Smoothed analysis of the condition numbers and growth factors of matrices. manuscript, 2002.
13. SPIELMAN, D., AND TENG, S. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)* (2001), pp. 296–305.
14. SPIELMAN, D., AND TENG, S. Smoothed analysis of property testing. manuscript, 2002.
15. ZHANG, L., DEVARAJAN, H., BASCH, J., AND INDYK, P. Probabilistic analysis for combinatorial functions of moving points. In *Proceedings of the 13th Annual ACM Symposium on Computational Geometry* (1997), pp. 442–444.

Kinetic Dictionaries: How to Shoot a Moving Target^{*}

Mark de Berg

Department of Computing Science, TU Eindhoven
P.O.Box 513, 5600 MB Eindhoven, The Netherlands.
`mdberg@win.tue.nl`

Abstract. A kinetic dictionary is a data structure for storing a set S of continuously moving points on the real line, such that at any time we can quickly determine for a given query point q whether $q \in S$. We study trade-offs between the worst-case query time in a kinetic dictionary and the total cost of maintaining it during the motions of the points.

1 Introduction

A *dictionary* is a data structure for storing a set S of elements—the elements are often called *keys*—such that one can quickly decide, for a given query element q , whether $q \in S$. Furthermore, the data structure should allow for insertions into and deletions from the set S . Often the keys come from a totally ordered universe. In this case one can view the keys as points on the real line. The dictionary is one of the most fundamental data structures in computer science, both from a theoretical point of view and from an application point of view. Hence, every algorithms book features various different possibilities to implement a dictionary: linked lists, (ordered) arrays, (balanced) binary search trees, hash tables, and so on.

In this paper we study a variant of the dictionary problem, where the keys come from a totally ordered universe but have continuously changing values. In other words, the set S is a set of points moving continuously on the real line. (Here ‘continuously’ does not mean that all points necessarily move all the time—this need not be the case—but rather that the motions are continuous.) This setting is motivated by a recent trend in the database community to study the indexing of moving objects—see for example [2,12,13,14] and the references therein. Also in the computational-geometry community, the study of data structures for moving objects has attracted a lot of attention recently—see for example [3,4,7,10] and the references therein.

The traditional approach to deal with moving objects is to use time-sampling: at regular time intervals one checks which objects have changed their position, and these objects are deleted from the data structure and re-inserted at their new positions. The problem with this approach is two-fold. First, it is hard to

^{*} Part of this research was done during a visit to Stanford University.

choose the right time interval: choosing it too large will mean that important ‘events’ are missed, so that the data structure will be incorrect for some time. Choosing the interval very small, on the other hand, will be very costly—and even in this case one is likely to miss important events, as these will usually occur at irregular times. Therefore the above-mentioned papers from computational geometry use so-called *kinetic data structures* (KDSs, for short), as introduced by Basch *et al.* in their seminal paper [7]. A KDS is a structure that maintains a certain ‘attribute’ of a set of continuously moving objects—the convex hull of moving points, for instance, or the closest distance among moving objects. It consists of two parts: a combinatorial description of the attribute, and a set of *certificates*—elementary tests on the input objects—with the property that as long as the outcome of the certificates does not change, the attribute does not change. In other words, the set of certificates forms a proof that the current combinatorial description of the attribute is still correct. The main idea behind KDSs is that, because the objects move continuously, the data structure only needs to be updated at certain *events*, namely when a certificate fails. It is assumed that each object follows a known trajectory—its *flight path*—so that one can compute the *failure time* of each certificate. When a certificate fails, the KDS and the set of certificates need to be updated. To know the next time any certificate fails, the failure times are stored in an event queue. The goal when designing a KDS is to make sure that there are not too many events, while also ensuring that the update time at an event is small—see the excellent survey by Guibas [9] for more background on KDSs and their analysis.

The problem we study is the following. Let S be a set of n points moving continuously on the real line, that is, the value of x_i at time t is a continuous function, $x_i(t)$, of time. We define $S(t) = \{x_1(t), \dots, x_n(t)\}$. When no confusion can arise, we often simply write S and x_i for $S(t)$ and $x_i(t)$. Our goal is to maintain a data structure for S such that, at any time t , we can quickly determine for a query point q whether $q \in S(t)$. Following the KDS framework, we require that the structure be correct—that is, that it returns the correct answer for any query—at all times. We call such a data structure a *kinetic dictionary*. (In this paper, we do not consider updates on the set S , so perhaps the name dictionary is a slight abuse of the terminology.) We assume that, at any time t , we can compute the current position $x_i(t)$ of any point x_i in $O(1)$ time. Note that t is not part of the query. This means that we do not allow queries in the past or in the future; we only allow queries about the current set S .

One possible implementation of a kinetic dictionary is to simply store all the points in a sorted array $D[1..n]$. Then, as long as the order of the points does not change, we can answer a query with a point q in $O(\log n)$ time by doing a binary search in D , using the current values $x_i(t)$ to guide the search. On the other hand, maintaining a sorted array means that whenever two points change their order, we need to update the structure. (In KDS terminology, the comparisons between consecutive points in the array are the certificates, and a swap of two elements is a certificate failure.) Even if the points have constant (but distinct) velocities, this approach may lead to $\Omega(n^2)$ updates. If there are only few queries

to be answered, then this is wasteful: one would rather have a structure with somewhat worse query time if that would mean that it has to be updated less frequently. This is the topic of our work: how often do we need to update a kinetic data structure to be able to guarantee a certain worst-case query time?

The kinetic dictionary problem was already studied by Agarwal *et al.* [2]. They described a data structure with $O(n^{1/2+\varepsilon})$ query time, for any¹ $\varepsilon > 0$, for the case where the points move linearly, that is, for the case where the points move with constant, but possibly different, velocities. Their structure needs to be updated $O(n)$ times. They also showed how to obtain trade-offs between query time and the number of updates for linear motions: for any parameter Q with $\log n \leq Q \leq n$, they have a structure that has $O(Q)$ query time, and that has to be updated $O(n^{2+\varepsilon}/Q^2)$ times. (Their solution has good I/O-complexity as well.)

The goal of our research is to study the fundamental complexity of the kinetic dictionary problem: What are the best trade-offs one can obtain between query time and number of updates? And how much knowledge of the motion do we need to obtain a certain trade-off? Our results for this are as follows.

First of all, since we are (also) interested in lower bounds, we need to establish a suitable ‘model of computation’. To this end we propose in the next section so-called *comparison graphs* as a model for kinetic dictionaries, and we define the cost of answering a query in this model, and the cost of updates.

We then continue to study the central question of our paper in this model: we want to bound the minimum total maintenance cost, under certain assumptions on the motions, when one has to guarantee worst-case query cost Q at all times. We start by describing in Section 3 a trivial solution with $O(n^2/Q)$ maintenance cost under the assumption that any pair of points changes $O(1)$ times. In Section 4 we then prove the following lower bound: any kinetic dictionary with worst-case query cost Q must have a total maintenance cost of $\Omega(n^2/Q^2)$ in the worst case, even if all points have fixed (but different) velocities. Note that the bounds of Agarwal *et al.* [2] almost match this lower bound. Their structure does not fit into our model, however. Hence, in Section 5 we show that their structure can be changed such that it fits into our model; the query time and the number of updates remains (almost) the same. Moreover, the result can be generalized such that it holds in a more general setting, namely when any two points exchange order at most once and, moreover, the complete motions are known in advance.

2 A Comparison-Based Model for Kinetic Dictionaries

Before we can prove lower bounds on the query cost and total maintenance cost in a kinetic dictionary, we must first establish a suitable ‘model of computation’: we must define the allowable operations and their cost. Let $S = \{x_1, \dots, x_n\}$ be a set of n points on the real line. Our model is comparison-based: the operations

¹ This type mean that one can fix any $\varepsilon > 0$, and then construct the data structure such that the query time is $O(n^{1/2+\varepsilon})$.

that we count are comparisons between two data points in S and between a data point and a query point. Note that we are not interested in a single-shot problem, but in maintaining a data structure to answer queries. Hence, comparisons can either be done when answering a query, or they can be done when constructing or updating the data structure. In the latter case, the comparisons can only be between data points, and the result has to be encoded in the data structure.

The idea of the lower bound will then be as follows. A query asks for a query point q whether $q \in S$. Suppose the answer to this query is negative. To be able to conclude this, we have to know for each $x \in S$ that $q \neq x$. This information can be obtained directly by doing a comparison between q and x ; this will incur a unit cost in the query time. It is also possible, however, to obtain this information indirectly. For example, if the information that $x' < x$ is encoded in the dictionary, and we find out that $q < x'$, then we can derive $q \neq x$. Thus by doing a single comparison with q , we may be able to derive the position of q relative to many points in S . This gain in query time has its cost, however: the additional information encoded in the dictionary has to be maintained. To summarize, comparisons needed to answer a query can either be done at the time of the query or they can be pre-computed and encoded in the dictionary; the first option will incur costs in the query time, the second option will incur maintenance costs.

In the remainder of this section we define our model more precisely.

The data structure. For simplicity we assume that all points in S are distinct. Of course there will be times when this assumption is invalid, otherwise the order would remain the same and the problem would not be interesting. But in our lower-bound arguments to be presented later, we will only argue about time instances where the points are distinct so this does not cause any serious problems.

A *comparison graph* for S is a directed graph $\mathcal{G}(S, A)$ with node set² S that has the following property: if $(x_i, x_j) \in A$ then $x_i < x_j$. The reverse is not true: $x_i < x_j$ does not imply that we must have $(x_i, x_j) \in A$. Note that a comparison graph is acyclic.

Query cost. Let q be a query point, and let $\mathcal{G} := \mathcal{G}(S, A)$ be a comparison graph. An *extended comparison graph* for q is a graph $\mathcal{G}_q^*$ with node set $S \cup \{q\}$ and arc set $A^* \supset A$. The arcs in A^* are of two types, *regular arcs* and *equality arcs*. They have the following property: if $(a, b) \in A^*$ is a regular arc then $a < b$, and if $(a, b) \in A^*$ is an equality arc then $a = b$. Note that for an equality arc (a, b) , either a or b must be the query point q , because we assumed that the points in S are distinct. A regular arc may or may not involve q . An extended comparison graph $\mathcal{G}_q^*$ for q *localizes* q if

- (i) it contains an equality arc, or
- (ii) for any point $x_i \in S$, there is a path in $\mathcal{G}_q^*$ from x_i to q or from q to x_i .

² In the sequel we often do not distinguish between a point in S and the corresponding node in $\mathcal{G}(S, A)$.

In the first case, we can conclude that $q \in S$, in the second case that $q \notin S$.

Given a comparison graph $\mathcal{G} = \mathcal{G}(S, A)$ for S and a query point q , we define the *query cost of q in $\mathcal{G}$* to be the minimum number of arcs we need to add to $\mathcal{G}_q = (S \cup \{q\}, A)$ to obtain an extended comparison graph $\mathcal{G}_q^*$ that localizes q . The (worst-case) query cost of $\mathcal{G}$ is the maximum query cost in $\mathcal{G}$ over all possible query points q . Our definition of query cost is justified by the following lemma.

Lemma 1. *Let $\mathcal{G}_q^*$ be an extended comparison graph with node set $S \cup \{q\}$ and arc set A^* . Suppose that $\mathcal{G}_q^*$ does not localize q . Then there are values for S and q that are consistent with the arcs in A^* such that $q \notin S$, and there are also values for S and q that are consistent with the arcs in A^* such that $q \in S$.*

Proof. If $\mathcal{G}_q^*$ does not localize q , then there are only regular arcs in A^* . Since $\mathcal{G}_q^*$ is acyclic, there exists a topological ordering of the nodes in the graph. By assigning each node the value corresponding to its position in the topological ordering, we obtain an assignment consistent with the arcs in A^* such that $q \notin S$.

Next we change the values of the nodes to obtain an assignment with $q \in S$. Consider the node $x_i \in S$ closest to q in the topological ordering—ties can be broken arbitrarily—such that there is no path in $\mathcal{G}_q^*$ between x_i and q . Because $\mathcal{G}_q^*$ does not localize q , such a node must exist. Now we make the value of x_i equal to the value of q . Assume that x_i was smaller than q in the original assignment; the case where x_i is larger can be handled similarly. Then the only arcs that might have become invalid by changing the value of x_i are arcs of the form (x_i, x_j) for some x_j that lies between the original value of x_i and q . Such a node x_j must have been closer to q in the topological ordering. By the choice of x_i , this implies that there is a path from x_j to q . But then the arc (x_i, x_j) cannot be in A^* , otherwise there would be a path from x_i to q , contradicting our assumptions. We can conclude that making x_i equal to q does not make any arcs invalid, so we have produced an assignment consistent with A^* such that $q \in S$. $\square$

Note that the query cost does not change when we restrict our attention to the transitive reduction [6] of the comparison graph. This is the subgraph consisting of all *non-redundant arcs*, that is, arcs that are not implied by other arcs because of transitivity. The transitive reduction of an acyclic graph is unique.

Our definition of query cost is quite weak, in the sense that it gives a lot of power to the query algorithm: the query algorithm is allowed to consult, free of charge, an oracle telling it which arcs to add to the graph (that is, which comparisons to do). This will only make our lower bounds stronger. When we discuss upper bounds, we will also show how to implement them in the real-RAM model.

Maintenance cost. We define the cost of updating a comparison graph to be equal to the number of new non-redundant arcs, that is, the number of non-redundant arcs in the new comparison graph that were not present in the transitive closure of the old comparison graph.

The rationale behind this is as follows. When using a kinetic data structure, one has to know when to update it. In our case, this is when some of the ordering information encoded in the kinetic dictionary is no longer valid. This happens exactly when the two points connected by an arc in the comparison graph change order; at that time, such an arc is necessarily non-redundant. (In the terminology of kinetic data structures, one would say that the arcs are the *certificates* of the data structures: as long as the certificates remain valid, the structure is guaranteed to be correct.) To know the next time such an event happens, the ‘failure times’ of these arcs are stored in an event queue. This means that when a new non-redundant arc appears, we have to compute its failure time and insert it into the event queue.

Examples. Next we have a look at some well known dictionary structures, and see how they relate to the model.

First, consider a binary search tree. This structure contains all ordering information, that is, the comparison graph corresponding to a binary search tree in the complete graph on S , with the arcs directed appropriately. The transitive reduction in this case is a single path containing all the nodes. A sorted array on the points has the same comparison graph as a binary search tree; sorted arrays and binary search trees are simply two different ways to implement a dictionary whose comparison graph is the complete graph.

The worst-case query cost of the complete graph is $O(1)$: when $q \notin S$ we need to add at most two regular arcs to localize any query point q (one from the predecessor of q in S and one to the successor of q in S), and when $q \in S$ a single equality arc suffices. This is less than the query time in a binary search tree, because we do not charge for the extra time needed to find out which two comparisons can do the job. In an actual implementation we would need to do a binary search to find the predecessor and successor of q in S , taking $O(\log n)$ time.

Our model does not apply to hash tables, since they are not comparison-based: with a hash function we can determine that $q \neq x$ without doing a comparison between q and x , so without knowing whether $q < x$ or $q > x$.

3 A Trivial Upper Bound

Suppose we want to have a kinetic dictionary whose worst-case query cost is Q at all times, for some $2 \leq Q \leq n$. A trivial way to achieve this is to partition the set S into $\lfloor Q/2 \rfloor$ subsets of size $O(n/Q)$ each, and to maintain each subset in a sorted array. Thus the transitive reduction of the corresponding comparison graph consists of $\lfloor Q/2 \rfloor$ paths. We need to add at most two arcs to localize a query point in a path, so the total query cost will be at most Q ; the actual query time in the real-RAM model would be $O(Q \log(n/Q))$. The total maintenance cost is linear in the number of pairs of points from the same subset changing order. If any pair of points changes order $O(1)$ times—which is true for instance when the motions are constant-degree algebraic functions—then this implies that

the maintenance cost of a single subset is $O((n/Q)^2)$. (This does not include the cost to insert and delete certificate failure times from the global event queue that any KDS needs to maintain. The extra cost for this is $O(\log n)$.) We get the following result.

Theorem 1. *Let S be a set of n points moving on the real line, and suppose that any pair of points changes order at most a constant number of times. For any Q with $2 \leq Q \leq n$, there is a comparison graph for S that has worst-case query cost Q and whose total maintenance cost is $O(n^2/Q)$.*

The comparison graph can be implemented such that the actual query time is $O(Q \log(n/Q))$, and the actual cost to process all the updates is $O(n^2/Q)$.

Our main interest lies in the question whether one can improve upon this trivial upper bound.

4 Lower Bounds for Linear Motions

We now turn our attention to lower bounds for kinetic dictionaries in the comparison-graph model. Our goal is to prove lower bounds regarding possible trade-offs between query cost and maintenance cost: what is the minimum amount of work we have to spend on updates if we want to guarantee cost Q always? Of course we will have to put some restrictions on the motions of the points, as otherwise we could always swap a pair of points that defines an arc in the comparison graph.

Here we consider a very limited scenario, where we only allow the points to move linearly. That is, all points have fixed (but possibly different) velocities. In this case we can show that any comparison graph that guarantees query cost at most Q must have a total update cost of $\Omega(n^2/Q^2)$. Our construction is based on the following lemma.

Lemma 2. *Let $\mathcal{G}$ be a comparison graph for a set S of n points, and let Q be a parameter with $1 \leq Q \leq n/2$. Suppose $\mathcal{G}$ has query cost Q . Then the subgraph induced by any subset of $2Q$ consecutive points from S contains at least Q non-redundant arcs.*

Proof. Let $x_1 < x_2 < \dots < x_n$ be the sorted sequence of points in S , and consider a subset $\{x_i, x_{i+1}, \dots, x_{i+2Q-1}\}$, for some i with $1 \leq i < n - 2Q$. Suppose we want to answer a query with a point q such that $x_i < q < x_{i+1}$. Note that $q \notin S$. In order to localize q , we need to add arcs to $\mathcal{G}$ such that for any point in S there is a path to or from q . In particular, there must be a path between q and each of the points $x_i, x_{i+1}, \dots, x_{i+2Q-1}$. Such a path cannot contain points x_j with $j < i$ or $j > i + 2Q - 1$. Hence, the subgraph induced by $\{x_i, x_{i+1}, \dots, x_{i+2Q-1}\} \cup \{q\}$ is connected (when viewed as an undirected graph) after the addition of the arcs by the query algorithm. This means that it contains at least $2Q$ non-redundant arcs. Since the number of arcs added by the query algorithm is bounded by Q by definition, $\mathcal{G}$ must have contained at least Q non-redundant arcs between points in $\{x_i, x_{i+1}, \dots, x_{i+2Q-1}\}$. $\square$

Lemma 2 implies that after the reversal of a group of $2Q$ consecutive points, the graph contains at least Q new non-redundant arcs. Hence, the reversal will induce a maintenance cost of at least Q .

We proceed by exhibiting a set of points moving linearly, such that there are many time instances where a subset of $2Q$ consecutive points completely reverses order. By the above lemma, this will then give a lower bound on the total maintenance cost.

The set of points is defined as follows. We can assume without loss of generality that $n/(2Q)$ is an integer. We have $n/(2Q)$ groups of points, each consisting of $2Q$ points. The points in S_i , the i -th group, are all coincident at $t = 0$: they all have value i at that time. (It is easy to remove this degeneracy.) The points are all moving linearly, so the trajectories of the points in the tx -plane are straight lines. In particular, in each group there is a point whose trajectory has slope j , for any integer j with $0 \leq j < 2Q$. More formally, we have groups $S_1, \dots, S_{n/(2Q)}$ defined as follows:

$$S_i := \{x_{ij}(t) : 0 \leq j < 2Q \text{ and } j \text{ integer}\}, \quad \text{where } x_{ij}(t) := i + jt.$$

Now consider a point (s, a) in the tx -plane, where a and s are integers with $n/(4Q) < a < n/(2Q)$ and $0 < s \leq n/(8Q^2)$. Then there are exactly $2Q$ trajectories passing through this point. To see this, consider a slope j with j integer and $0 \leq j < 2Q$. Then the line $x(t) = jt + (a - sj)$ passes through (s, a) and the restrictions on a and s ensure that $a - sj$ is an integer with $0 \leq a - sj < n/(2Q)$, so this line is one of the trajectories. We conclude that there are $\Omega(n^2/Q^3)$ points (s, a) in the tx -plane such that $2Q$ trajectories meet at (s, a) .

Theorem 2. *There is a set S of n points, each moving with constant velocity on the real line, such that, in the comparison-graph model, any kinetic dictionary for S with worst-case query cost Q has total update cost $\Omega(n^2/Q^2)$.*

Proof. In the construction described above there are $\Omega(n^2/Q^3)$ points (s, a) in the ty -plane at which $2Q$ points meet. These $2Q$ points are consecutive just before and just after time s , and their order completely reverses at time s . It follows from Lemma 2 that the reversal of one such group forces the comparison graph to create at least Q new non-redundant arcs within the group. Hence, the total update cost is $\Omega(n^2/Q^2)$. $\square$

5 Upper Bounds for Pseudo-Linear Motions

In this section we show that the lower bounds of the previous section are almost tight if any pair of points swaps at most once—we call such motions *pseudo-linear motions*—and, additionally, the motions are known in advance. A similar result has already been shown for linear motions by Agarwal *et al.* [2].

We proceed as follows. Suppose for now that the motions are linear. Draw the trajectories of the points in the ty -plane. This way we obtain a set of n lines.

A query with a point q at time t now amounts to checking whether the point (t, q) lies on any of the lines. Using a standard 2-dimensional range-searching structure, we can answer such queries in time $O(Q)$ with a data structure using $O((n^2/Q^2) \log^3 n)$ storage [1]. This structure is more powerful than required, since it allows for queries in the past and in the future; in fact, no updates are necessary during the motions. Unfortunately, it uses a super-linear amount of storage. Moreover, the structure does not fit into our model—see the proof of Lemma 5. Next we show how to transform it to a comparison graph. Our approach is similar to the approach of Agarwal *et al.*, who also describe a solution with linear space; our task is to change their solution so that it fits into our model.

The 2-dimensional range-searching structure has two ingredients: a structure with logarithmic query time but quadratic storage, and a structure with roughly $O(\sqrt{n})$ query time and linear storage. Next we describe these two ingredients in our kinetic-dictionary setting.

Lemma 3. *Let S be a set of n points moving on the y -axis. There is a comparison graph for S that has worst-case query cost $O(1)$ and needs to be updated only when two points exchange order. The update cost at such an event is $O(1)$.*

The comparison graph can be implemented such that the actual query time is $O(\log n)$, and the actual cost to process an update is $O(1)$.

Proof. Apply Theorem 3 with $Q = 2$. □

To get a structure with a near-linear number of updates we need the following result.

Lemma 4. *Let S be a set of n points moving on the y -axis such that any pair swaps at most once, and let r be a parameter with $1 \leq r \leq n$. There exists a partitioning of S into r disjoint subsets $S_1, \dots, S_r$ of size between n/r and $2n/r$ such that the following holds: at any point in time and for any query point q , we have that $\min(S_i) \leq q \leq \max(S_i)$ for at most $O(\sqrt{r})$ sets S_i .*

Proof. This follows from standard techniques. For completeness, we describe how the partitioning can be obtained.

First, assume the motions are linear. Thus the trajectories of the points are straight lines in the ty -plane. Dualize [8] these lines to obtain a set S^* of n points, and construct a fine simplicial partition of size r of the set of points. This is a collection of r pairs (S_i^*, Δ_i) where the S_i^* form a partition of S^* into subsets of size between n/r and $2n/r$ and each Δ_i is a triangle containing S_i^* . Matoušek[11] has shown that a fine simplicial partition exists with the following property: any line crosses at most $O(\sqrt{r})$ triangles Δ_i . Translated back into primal space, this is exactly the property that we need.

If the motions are pseudo-linear motions, we proceed as follows. Again, we consider the trajectories in the ty -plane. By definition, these are pseudo-lines, that is, monotone curves with each pair crossing at most once. By combining the construction of Matoušek for simplicial partitions with the techniques of Agarwal and Sharir [5] for dualizing pseudo-lines, we can again get a partition with the desired properties. □

We can now describe the kinetic dictionary with a near-linear number of updates.

Lemma 5. *Let S be a set of n points moving on the y -axis such that any pair swaps at most once, and such that the motions are completely known in advance. Then, for any $\varepsilon > 0$, there is a comparison graph for S that has worst-case query complexity $O(n^{1/2+\varepsilon})$ and needs to be updated $O(n \log^2 n)$ times.*

The comparison graph can be implemented such that the actual query time is $O(n^{1/2+\varepsilon})$, and the actual cost to process an update is $O(\log^2 n)$.

Proof. The comparison graph is constructed similarly to the way in which a 2-dimensional range-searching structure is constructed, as follows. The comparison graph is constructed recursively. Let r be a sufficiently large constant.

- We maintain two kinetic tournaments [7] for each subset S , one to maintain $\min(S)$ and one to maintain $\max(S)$. A kinetic tournament to maintain the minimum, say, is a balanced binary tree whose leaves are the points in S , and where every internal node ν stores the minimum of $S(\nu)$, the subset of points stored in the subtree rooted at ν . The arcs contributed by the kinetic tournament to the comparison graph are arcs between the points $\min(S(\nu))$ and $\min(S(\mu))$ stored at sibling nodes ν and μ .
- Next, we construct a partitioning of S into r subsets, as in Lemma 4. Each set S_i is stored recursively in a comparison graph.

Note: If we did not require the structure to be a comparison graph, we would store the triangles of the simplicial partition in the dual plane (see the proof of Lemma 4) directly. This would enable us to check whether we would have to visit a subtree, and we would not need the kinetic tournaments. The use of the kinetic tournaments is thus where we depart from the structure of Agarwal *et al.* [2].

- Finally, for each S_i we add an arc to the comparison graph which goes from $\max(S_i)$ to $\max(S)$, and an arc from $\min(S)$ to $\min(S_i)$. If these points happen to be the same, the arc is omitted.

Note: These arcs are not needed by the query algorithm, but they are required to ensure that, after performing the query algorithm, we have localized the query point according to the definition of Section 2.

The recursive construction ends when the size of the set drops below some fixed constant.

The resulting comparison graph is, of course, simply a directed graph on S . It is convenient, however, to follow the construction algorithm and think of the graph as a hierarchical structure. For a query point q , we can construct an extended comparison graph that localizes q , as follows. We compare q to $\min(S)$ and $\max(S)$, and we add the resulting arcs to the graph. If one of them happens to be an equality arc, we have localized q and can report that $q \in S$. If $q < \min(S)$ or $q > \max(S)$, then we can stop (with this ‘recursive call’) and conclude that q is not in the subtree rooted at ν . Otherwise, we recursively localize q in the comparison graphs of the children of the root. When we are at a leaf, we simply compare q to all the points stored there, and add the resulting arcs to the graph. It is easily seen that this procedure correctly localizes q . Furthermore, $A(n)$,

the total number of arcs added, satisfies the same recurrence one gets for range searching with a partition tree, namely

$$A(n) = O(\sqrt{r}) \cdot A(2n/r) + O(1).$$

For any $\varepsilon > 0$, we can choose r sufficiently large such that the solution of the recurrence is $O(n^{1/2+\varepsilon})$.

An update occurs when two points connected by an arc exchange order. It can be shown [7] that for (pseudo-)linear motions a kinetic tournament on a subset $S' \subset S$ processes $O(|S'| \log |S'|)$ events, which implies that the total number of events is $O(n \log^2 n)$. Updating the comparison graph at such an event means that we have to update the kinetic tournaments where the event occurs. A single swap can occur simultaneously in $O(\log n)$ tournaments, and updating one tournament tree has cost $O(\log n)$. Hence, the total update cost is $O(\log^2 n)$. Within this time we can also update the arcs between the maxima (and minima) of a node and its parent where needed.

Implementing this structure to achieve the same actual bounds is straightforward. $\square$

Theorem 3. *Let S be a set of n points moving with constant velocity on the y -axis, and let Q be a parameter with $2 \leq Q \leq n$. There is a comparison graph for S that has worst-case query complexity $O(Q)$ and needs to be updated $O(n^{2+\varepsilon}/Q^2)$ times. The cost of an update is $O(\log^2 n)$.*

The comparison graph can be implemented such that the actual query time is $O(Q \log(n/Q))$, and the actual cost to process all the updates is $O(n^{2+\varepsilon}/Q^2)$.

Proof. This can be done by combining the two structures described above, in the standard way: start with the recursive construction of Lemma 5, and switch to the structure of Lemma 3 when the number of points becomes small enough. More precisely, we switch when the number of points drops below $n/Q^{2-4\varepsilon}$, which gives the desired bounds. $\square$

6 Discussion

In this paper we discussed the problem of maintaining a dictionary on a set of points moving continuously on the real line. We defined a model for such kinetic dictionaries—the comparison-graph model—and in this model we studied trade-offs between the worst-case query cost and the total maintenance cost of kinetic dictionaries. In particular, we gave a trivial solution with query cost Q whose total maintenance cost is $O(n^2/Q)$, assuming any pair of points changes order $O(1)$ time, and we proved that $\Omega(n^2/Q^2)$ is a lower bound on the total maintenance cost of any kinetic dictionary with query time Q , even when each point has a fixed velocity. We also showed that the lower bound is almost tight if the motions are known in advance and any two points swap at most once.

The most challenging open problem is what happens when the motions are not known in advance, or when a pair of points can change order some constant (greater than one) number of times. Can one beat the trivial solution for this case?

Acknowledgement. I would like to thank Julien Basch and Otfried Cheong for stimulating discussions, and Jeff Erickson for pointing out that the upper bound for linear motions also works for the case of pseudo-linear motions.

References

1. P.K. Agarwal. Range searching. In: J.E. Goodman and J. O'Rourke (eds.), *Handbook of Discrete and Computational Geometry*, CRC Press, pages 575–598, 1997.
2. P.K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proc. Annu. ACM Sympos. Principles Database Syst.*, pages 175–186, 2000.
3. P.K. Agarwal, J. Basch, M. de Berg, L.J. Guibas, and J. Hershberger. Lower bounds for kinetic planar subdivisions. *Discrete Comput. Geom.* 24:721–733 (2000).
4. P.K. Agarwal and S. Har-Peled. Maintaining approximate extent measures of moving points. In *Proc. 12th ACM-SIAM Symp. Discrete Algorithms*, 2001.
5. P.K. Agarwal and M. Sharir. Pseudoline arrangements: duality, algorithms, and applications. In *Proc. 13th ACM-SIAM Symp. Discrete Algorithms*, 2002.
6. J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, 2001.
7. J. Basch, L.J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Alg.* 31:1–28 (1999).
8. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, 1997.
9. L.J. Guibas. Kinetic data structures—a state-of-the-art report. In *Proc. 3rd Workshop Algorithmic Found. Robot.*, pages 191–209, 1998.
10. D. Kirkpatrick and B. Speckmann. Kinetic maintenance of context-sensitive hierarchical representations of disjoint simple polygons. In *Proc. 18th Annu. ACM Symp. Comput. Geom.*, pages 179–188, 2002.
11. J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.* 8:315–334 (1992).
12. D. Pfoser, C.J. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proc. 26th Int. Conf. Very Large Databases*, pages 395–406, 2000.
13. S. Šaltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez. Indexing the positions of continuously moving objects. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, pages 331–342, 2000.
14. O. Wolfson, A.P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, pages 257–287, 1999.

Deterministic Rendezvous in Graphs

Anders Dessmark¹, Pierre Fraigniaud², and Andrzej Pelc³

¹ Dept. of Computer Science, Lund Univ., Box 118, S-22100 Lund, Sweden.

andersd@cs.lth.se

² CNRS, LRI, Univ. Paris Sud, 91405 Orsay, France. <http://www.lri.fr/~pierre>

³ Dép. d'Informatique, Univ. du Québec en Outaouais, Hull, Québec J8X 3X7,
Canada. pelc@uqo.ca

Abstract. Two mobile agents having distinct identifiers and located in nodes of an unknown anonymous connected graph, have to meet at some node of the graph. We present fast *deterministic* algorithms for this rendezvous problem.

1 Introduction

Two mobile agents located in nodes of a network, modeled as an undirected connected graph, have to meet at some node of the graph. This task is known as the rendezvous problem in graphs, and in this paper we seek efficient deterministic algorithms to solve it. If nodes of the graph are labeled then agents can decide to meet at a predetermined node and the rendezvous problem reduces to graph exploration. However, in many applications, when rendezvous is needed in an unknown environment, such unique labeling of nodes may not be available, or limited sensory capabilities of the agents may prevent them from perceiving such labels. Hence it is important to be able to program the agents to explore *anonymous* graphs, i.e., graphs without unique labeling of nodes. Clearly, the agents have to be able to *locally* distinguish ports at a node: otherwise, an agent may even be unable to visit all neighbors of a node of degree 3 (after visiting the second neighbor, the agent cannot distinguish the port leading to the first visited neighbor from that leading to the unvisited one). Consequently, agents initially located at two nodes of degree 3, might never be able to meet. Hence we make a natural assumption that all ports at a node are locally labeled $1, \dots, d$, where d is the degree of the node. No coherence between those local labelings is assumed. We also do not assume any knowledge of the topology of the graph or of its size. Likewise, agents are unaware of the distance separating them.

Agents move in synchronous rounds. In every round, an agent may either remain in the same node or move to an adjacent node. We consider two scenarios: *simultaneous startup*, when both agents start executing the algorithm at the same time, and *arbitrary startup*, when starting times are arbitrarily decided by the adversary. In the former case, agents know that starting times are the same, while in the latter case, they are not aware of the difference between starting times, and each of them starts executing the rendezvous algorithm and counting rounds since its own startup. The agent who starts earlier and happens to visit

the starting node of the later agent *before* the startup of this later agent, is not aware of this fact, i.e., we assume that agents are created at their startup time and not waiting in the node before it.

An agent, currently located at a node, does not know the other endpoints of yet unexplored incident edges. If the agent decides to traverse such a new edge, the choice of the actual edge belongs to the adversary, as we are interested in the worst-case performance. We assume that, if agents get to the same node in the same round, they become aware of it and rendezvous is achieved. However, if agents cross each other along an edge (moving in the same round along the same edge in opposite directions) they do not notice this fact. In particular, rendezvous is not possible in the middle of an edge. The *time* used by a rendezvous algorithm, for a given initial location of agents in a graph, is the worst-case number of rounds since the startup of the later agent until rendezvous is achieved, where the worst case is taken over all adversary decisions, whenever an agent decides to explore a new edge adjacent to a currently visited node, and over all possible startup times (decided by the adversary), in case of the arbitrary startup scenario.

If agents are identical, i.e., they do not have distinct identifiers, and execute the same algorithm, then deterministic rendezvous is impossible even in the simplest case when the graph consists of two nodes joined by an edge, agents are initially located at both ends of it and start simultaneously: in every round both agents will either stay in different nodes or will both move to different nodes, thus they will never meet. Hence we assume that agents have distinct identifiers, called labels, which are two different integers written as binary strings starting with 1, and that every agent knows its own label. Now, if *both* agents knew *both* labels, the problem can be again reduced to that of graph exploration: the agent with smaller label does not move, and the other agent searches the graph until it finds it. However, the assumption that agents know each other may often be unrealistic: agents may be created in different parts of the graph in a distributed fashion, oblivious of each other. Hence we assume that each agent knows its own label but does not know the label of the other. The only initial input of a (deterministic) rendezvous algorithm executed by an agent is the agent's label. During the execution of the algorithm, an agent learns the local port number by which it enters a node and the degree of the node.

In this setting, it is not even obvious that (deterministic) rendezvous is at all possible. Of course, if the graph has a distinguished node, e.g., a unique node of a given degree, agents could decide to meet at this node, and hence rendezvous would be reduced to exploration (note that an agent visiting a node becomes aware of its degree). However, a graph may not have such a node, or its existence may be unknown and hence impossible to use in the algorithm. For example, it does not seem obvious *a priori* if rendezvous can be achieved in a ring.

The following are the two main questions guiding our research:

Q1. Is rendezvous feasible in arbitrary graphs?

Q2. If so, can it be performed efficiently, i.e., in time polynomial in the number n of nodes, in the difference τ between startup times and in labels L_1, L_2 of the agents (or even polynomial in n, τ and $\log L_1, \log L_2$)?

Our results. We start by introducing the problem in the relatively simple case of rendezvous in trees. We show that rendezvous can be completed in time $O(n + \log l)$ on any n -node tree, where l is the smaller of the two labels, even with arbitrary startup. We also show that for some trees this complexity cannot be improved, even with simultaneous startup. Trees are, however, a special case from the point of view of the rendezvous problem, as any tree has either a central node or a central edge, which facilitates the meeting (incidentally, the possibility of the second case makes rendezvous not quite trivial, even in trees). As soon as the graph contains cycles, the technique which we use for trees cannot be applied. Hence it is natural to concentrate on the simplest class of such graphs, i.e., rings. We prove that, with simultaneous startup, optimal time of rendezvous on any ring is $\Theta(D \log l)$, where D is the initial distance between agents. We construct an algorithm achieving rendezvous with this complexity and show that, for any distance D , it cannot be improved. With arbitrary startup, $\Omega(n + D \log l)$ is a lower bound on the time required for rendezvous on an n -node ring. Under this scenario, we show two rendezvous algorithms for the ring: an algorithm working in time $O(n \log l)$, for known n , and an algorithm polynomial in n , l and the difference τ between startup times, if n is unknown. For arbitrary graphs, our main contribution is a general feasibility result: rendezvous can be accomplished on arbitrary connected graphs, even with arbitrary startup. If simultaneous startup is assumed, we construct a generic rendezvous algorithm, working for all connected graphs, which is optimal for the class of graphs of bounded degree, if the initial distance between agents is bounded.

Related work. The rendezvous problem has been introduced in [16]. The vast literature on rendezvous (see the book [3] for a complete discussion and more references) can be divided into two classes: papers considering the geometric scenario (rendezvous in the line, see, e.g., [10,11,13], or in the plane, see, e.g., [8, 9]), and those discussing rendezvous in graphs, e.g., [1,4]. Most of the papers, e.g., [1,2,6,10] consider the probabilistic scenario: inputs and/or rendezvous strategies are random. A natural extension of the rendezvous problem is that of gathering [12,15,17], when more than 2 agents have to meet in one location. To the best of our knowledge, the present paper is the first to consider deterministic rendezvous in unlabeled graphs assuming that each agent knows only its own identity.

Terminology and notation. Labels of agents are denoted by L_1 and L_2 . The agent with label L_i is called agent i . (An agent does not know its number, only its label). Labels are distinct integers represented as binary strings starting with 1. l denotes the smaller of the two labels. The difference between startup times of the agents is denoted by τ . The agent with earlier startup is called the *earlier* agent and the other agent is called the *later* agent. In the case of simultaneous startup, the earlier agent is defined as agent 1. (An agent does not know if it is earlier or later). We use the word “graph” to mean a simple undirected connected graph. n denotes the number of nodes in the graph, Δ the maximum degree, and D the distance between initial positions of agents.

2 Rendezvous in Trees

We introduce the rendezvous problem in the relatively simple case of trees. In this section we assume that agents know that they are in a tree, although they know neither the topology of the tree nor its size. Trees have a convenient feature from the point of view of rendezvous. Every tree has either a *central node*, defined as the unique node minimizing the distance from the farthest leaf, or a *central edge*, defined as the edge joining the only two such nodes. This suggests an idea for a natural rendezvous algorithm, even for arbitrary startup: explore the tree, find the central node or the central edge, and try to meet there. Exploring the tree is not a problem: an agent can perform DFS, keeping a stack for used port numbers. At the end of the exploration, the agent has a map of the tree, can identify the central node or the central edge, and can find its way either to the central node or to one endpoint of the central edge, in the latter case knowing which port corresponds to the central edge. In the first case, rendezvous is accomplished after the later agent gets to the central node. In the second case, the rendezvous problem in trees can be reduced to rendezvous on graph K_2 consisting of two nodes joined by an edge. We now show a procedure for rendezvous in this simplest graph.

The Procedure Extend-Labels, presented below, performs a rendezvous on graph K_2 in the model with arbitrary startup. The procedure is formulated for an agent with label L . Enumerate the bits of the binary representation of L from left to right, i.e., starting with the most significant bit. The actions taken by agents are either *move* (i.e., traverse the edge) or *stay* (i.e., remain in place for one round). Rounds are counted from the starting round of the agent. Intuitively, the behavior of the agent is the following. First, transform label L into the string L^* by writing bits 10 and then writing twice every bit of L . Then repeat indefinitely string L^* , forming an infinite binary string. The agent moves (stays) in round i , if the i th position of this infinite string is 1 (0). Below we give a more formal description of the algorithm.

Procedure Extend-Labels

In round 1 *move*.

In round 2 *stay*.

In rounds $3 \leq i \leq 2\lfloor \log L \rfloor + 4$, *move* if bit $\lceil (i-2)/2 \rceil$ of L is 1, otherwise *stay*.

In rounds $i > 2\lfloor \log L \rfloor + 4$ behave as for round $1 + (i-1 \bmod 2\lfloor \log L \rfloor + 4)$.

The following illustrates the execution of Procedure Extend-Labels for label 101:

1	0	1	1	0	0	1	1	1	0	1	1	0	0	1	1	1	0	...

Theorem 1. *Procedure Extend-Labels performs rendezvous on graph K_2 in at most $2\lfloor \log l \rfloor + 6$ rounds.*

Proof. Assume, without loss of generality, that agent 1 starts not later than agent 2. Rounds are counted from the startup of agent 2. The proof is divided into four cases.

Case 1. τ is odd. In this case, either agent 1 stays in the second round or a rendezvous is accomplished in one of the first two rounds. If agent 1 stays in the

second round, since this is an odd round for this agent, it also stays in the third round. In the third round, however, agent 2 moves, as its action corresponds to the most significant bit of L_2 , which is 1. Thus rendezvous is accomplished no later than in round 3.

Case 2. τ is even and not divisible by $2\lfloor \log L_1 \rfloor + 4$. In this case, the actions of agent 1 in the first two rounds are decided by the same bit in L_1 . Thus, the actions of the two agents will be different in one of the rounds and a rendezvous is accomplished no later than in round 2.

Case 3. τ is even and divisible by $2\lfloor \log L_1 \rfloor + 4$, and $\lfloor \log L_1 \rfloor = \lfloor \log L_2 \rfloor$. In this case, at least one bit must be different in both labels. Let b be the position of this bit. In round $2b + 1$ the behavior of the two agents is different, and a rendezvous is accomplished. Thus, rendezvous is accomplished no later than in round $2\lfloor \log L_1 \rfloor + 3$.

Case 4. τ is even and divisible by $2\lfloor \log L_1 \rfloor + 4$, and $\lfloor \log L_1 \rfloor \neq \lfloor \log L_2 \rfloor$. In this case, the actions of the agent with the smaller label are different in rounds $2\lfloor \log l \rfloor + 5$ and $2\lfloor \log l \rfloor + 6$, while the other agent performs the same action. This results in a rendezvous no later than in round $2\lfloor \log l \rfloor + 6$. $\square$

We can now formulate the following general algorithm for rendezvous in trees.

Algorithm Rendezvous-in-Trees

- (1) Explore the tree
- (2) **if** there is a central node **then** go to this node and stay;
 else go to one endpoint of the central edge;
 execute Procedure Extend-Labels;

Theorem 2. *Algorithm Rendezvous-in-Trees performs rendezvous on any n -node tree in $O(n + \log l)$ rounds. On the other hand, there exist n -node trees on which any rendezvous algorithm requires time $\Omega(n + \log l)$, even with simultaneous startup.*

Proof. The upper bound follows the fact that exploration of an n -node tree can be done in time $O(n)$. Now, consider an n -node tree, with $n = 2k$, consisting of two stars of degree $k - 1$ whose centers are joined by an edge. Suppose that agents are initially located in centers of these stars and start simultaneously. The adversary can prevent an agent from finding the edge joining their initial positions for $2(k - 1) \in \Omega(n)$ rounds. (After each unsuccessful attempt, the agent has to get back to its starting node.) This proves the lower bound $\Omega(n)$ on the time of rendezvous. In order to complete the proof, it is enough to show the lower bound $\Omega(\log l)$. We prove it in the simpler case of the two-node graph. (This also proves that Procedure Extend-Label is optimal for the two-node graph.) It is easy to extend the argument for our tree.

For any integer $x > 2$ and any rendezvous algorithm working in $t < x - 1$ rounds, we show two labels L_1 and L_2 of length x , such that the algorithm fails if agents placed on both ends of the edge have these labels. Let S_i be the binary sequence of length t describing the move/stay behavior of agent i (if the agent moves in round r , the r th bit of its sequence is 1, otherwise it is 0). Since S_i is a function of L_i , and there are only $2^t < 2^{x-1}$ possible sequences S_i , it follows that

there exist two distinct labels L_1 and L_2 of length x , such that $S_1 = S_2$. Pick those two labels. During the first t rounds, agents exhibit the same move/stay behavior, and hence they cannot meet. $\square$

3 Rendezvous in Rings

In this section we assume that agents know that the underlying graph is a ring, although, in general, they do not know its size.

3.1 Simultaneous Startup

We provide an algorithm that performs rendezvous on a ring in the simultaneous startup model, and prove that it works in an asymptotically optimal number of rounds. In order to simplify the presentation, we first propose two algorithms that work only under certain additional conditions, and then show how to merge these algorithms into the final algorithm, working in all cases. Our first algorithm, *Similar-Length-Labels*, works under the condition that the lengths of the two labels are similar, more precisely, $\lfloor \log \log L_1 \rfloor = \lfloor \log \log L_2 \rfloor$. Let the extended label L_i^* be a sequence of bits of length $2^{\lfloor \log \log L_i \rfloor + 1}$, consisting of the binary representation of label L_i preceded by a (possibly empty) string of zeros. For example, the label 15 corresponds to the binary sequence 1111, while the label 16 corresponds to 00010000. The algorithm is formulated for an agent with label L and corresponding extended label L^* . Let $m = 2^{\lfloor \log \log L \rfloor + 1}$ be the length of L^* . The algorithm works in stages numbered $1, 2, \dots$ until rendezvous is accomplished. Stage s consists of m phases, each of which has 2^{s-1} rounds.

Algorithm Similar-Length-Labels

In phase b of stage s do:

if bit b of L^* is 1 then

- (1) move for 2^{s-1} rounds in an arbitrary direction from the starting node;
- (2) move for 2^s rounds in the opposite direction;
- (3) go back to the starting node

else stay for 2^{s+1} rounds.

Lemma 1. *Algorithm Similar-Length-Labels performs rendezvous in $O(D \log l)$ rounds on a ring, if $\lfloor \log \log L_1 \rfloor = \lfloor \log \log L_2 \rfloor$.*

Proof. If $\lfloor \log \log L_1 \rfloor = \lfloor \log \log L_2 \rfloor$, the lengths of the extended labels of both agents are equal, and therefore any phase b of any stage s of agent 1 starts and ends at the same time as for agent 2. Since $L_1 \neq L_2$, one of the agents is moving while the other is staying in at least one phase b of every stage. During stage s , every node at distance 2^{s-1} from the starting point of the agent will be visited. Thus, in phase b of stage s , where s is the smallest integer such that $2^{s-1} \geq D$, the agent that moves in this phase meets the agent that stays in this phase. The number of rounds in this stage is $O(D \log l)$, which also dominates the sum of rounds in all previous stages. $\square$

Our second algorithm, Different-Length-Labels, works under the condition that $\lfloor \log \log L_1 \rfloor \neq \lfloor \log \log L_2 \rfloor$. Let the *activity number* b_i for agent i be 1 if $L_i = 1$, and $2 + \lfloor \log \log L_i \rfloor$ otherwise. The algorithm is formulated for an agent with label L and activity number b . The algorithm works in stages. Stage s consists of s phases. Phase p of any stage consists of 2^{p+1} rounds.

Algorithm Different-Length-Labels

In stage $s < b$, stay;

In stage $s \geq b$, stay in phases $p \neq s - b + 1$;

In phase $p = s - b + 1$ of stage $s \geq b$, move for 2^{p-1} rounds in an arbitrary direction, move in the opposite direction for 2^p rounds, and go back to the starting node.

Lemma 2. *Algorithm Different-Length-Labels performs a rendezvous in $O(D \log l)$ rounds, if $\lfloor \log \log L_1 \rfloor \neq \lfloor \log \log L_2 \rfloor$.*

Proof. If $\lfloor \log \log L_1 \rfloor \neq \lfloor \log \log L_2 \rfloor$, the agents have different activity numbers. Assume, without loss of generality, that $l = L_1$. Hence $b_1 < b_2$. In stage $s \geq b_1$, agent 1 visits every node within distance 2^{s-b_1} from its starting node (in phase $s - b_1 + 1$). Hence, rendezvous is accomplished in stage s , where s is the smallest integer such that $2^{s-b_1} \geq D$, i.e., $s = b_1 + \lceil \log D \rceil$. Stage s consists of $O(2^s)$ rounds and dominates the sum of rounds in all previous phases. The required number of rounds is thus $O(2^{b_1 + \log D}) = O(D \log l)$. $\square$

We now show how to combine Algorithm Similar-Length-Labels with Algorithm Different-Length-Labels into an algorithm that works for entirely unknown labels. The idea is to interleave rounds where Algorithm Similar-Length-Labels is performed with rounds where Algorithm Different-Length-Labels is performed. However, this must be done with some care, as an agent cannot successfully switch algorithms when away from its starting node. The solution is to assign slices of time of increasing size to the algorithms. At the beginning of a phase of each of the algorithms, the agent is at its starting node. If it can complete the given phase of this algorithm before the end of the current time slice, it does so. Otherwise it waits (at its starting node) until the beginning of the next time slice (devoted to the execution of the other algorithm), and then proceeds with the execution of the halted phase in the following time slice. (Note that, while one agent remains idle till the end of a time slice, the other agent might be active, if Algorithm Similar-Length-Labels is executed and the label lengths are in different ranges.)

It only remains to specify the sequence of time slices. Let time slice t consist of 2^{t+1} rounds (shorter slices than 4 rounds are pointless). It is now enough to notice that the phases up for execution during a time slice will never have more rounds than the total number of rounds in the slice. As a phase of an algorithm has never more than twice the number of rounds of the preceding phase, at least a constant fraction of every time slice is actually utilized by the algorithm. Exactly one of the algorithms has its condition fulfilled by the labels, and this algorithm accomplishes a rendezvous in $O(D \log n)$ rounds, while the other algorithm has been assigned at most twice as many rounds in total.

Theorem 3. *In the simultaneous startup model, the minimum time of rendezvous in the ring is $\Theta(D \log l)$.*

Proof. The upper bound has been shown above. For the lower bound, if $D = 1$, then the lower bound proof from the previous section is easy to modify for the ring. Thus assume that $D > 1$. We actually prove a lower bound for the weaker task *cross-or-meet* in which the two agents have either to meet at the same node, or to simultaneously traverse an edge in the two opposite directions. Clearly, an algorithm solving cross-or-meet in r rounds for two agents at distance D solves cross-or-meet in at most r rounds for two agents at distance $D - 1$. Thus we assume without loss of generality that D is even. Define an infinite sequence of consecutive segments of the ring, of $D/2$ vertices each, starting clockwise from an arbitrary node in the ring. Note that the starting nodes of the agents are located in two different segments, with one or two segments in between. Note also that the two agents have the same position within their segments. Divide all rounds into periods of $D/2$ rounds each, with the first round as the first round of the first period. During any period, an agent can only visit nodes of the segment where it starts the period and the two adjacent segments.

Suppose that port numbers (fixed by the adversary at every node) yield an orientation of the ring, i.e., for any node v , the left neighbor of the right neighbor of v is v . The behavior of an agent with label L , running algorithm A , yields the following sequence of integers in $\{-1, 0, 1\}$, called the *behavior code*. The t th term of the behavior code of an agent is -1 if the agent ends time period t in the segment to the left of where it began the period, 1 if it ends to the right and 0 if it ends in the segment in which it began the period. In view of the orientation of the ring, the behavior of an agent, and hence its behavior code, depends only on the label of the agent. Note that two agents with the same behavior code of length x , cannot accomplish cross-or-meet during the first x periods, if they start separated by at least one segment: even though they may enter the same segment during the period, there is insufficient time to visit the same node or the same edge.

Assume that there exists an algorithm A which accomplishes cross-or-meet in $Dy/6$ rounds. This time corresponds to at most $y/2$ periods. There are only $3^{y/2} < 2^y$ behavior codes of length $y/2$. Hence it is possible to pick two distinct labels L_1 and L_2 not greater than 2^y , for which the behavior code is the same. For these labels algorithm A does not accomplish cross-or-meet in $Dy/6$ rounds. This contradiction implies that any cross-or-meet algorithm, and hence any rendezvous algorithm, requires time $\Omega(D \log l)$. $\square$

3.2 Arbitrary Startup

We begin by observing that, unlike in the case of simultaneous startup, $\Omega(n)$ is a natural lower bound for rendezvous time in an n -node ring, if startup is arbitrary, even for bounded distance D between starting nodes of the agents. Indeed, since starting nodes can be antipodal, each of the agents must at some point travel at distance at least $n/4$ from its starting node, unless he meets the other agent before. Suppose that the later agent starts at the time when

the earlier agent is at distance $n/4$ from its starting node v . The distance D between the starting node of the later agent and v can be any number from 1 to an , where $a < 1/4$. Then rendezvous requires time $\Omega(n)$ (counting, as usual, from the startup of the later agent), since at the startup of the later agent the distance between agents is $\Omega(n)$. On the other hand, the lower bound $\Omega(D \log l)$ from the previous subsection is still valid, since the adversary may also choose simultaneous startup. Hence we have:

Proposition 1. *In the arbitrary startup model, the minimum time of rendezvous in the n -node ring is $\Omega(n + D \log l)$.*

We now turn attention to upper bounds on the time of rendezvous in the ring with arbitrary startup. Our next result uses the additional assumption that the size n of the ring is known to the agents. The idea is to modify Procedure Extend-Labels. Every round in Procedure Extend-Labels is replaced by $2n$ rounds: the agent stays, respectively moves in one (arbitrary) direction, for this amount of time. Recall that in the Procedure Extend-Labels the actions of the two agents differ in round $2\lfloor \log l \rfloor + 6$ at the latest (counting from the startup of the later agent). In the modified procedure, time segments of activity or passivity, lasting $2n$ rounds, need not be synchronized between the two agents (if τ is not a multiple of $2n$) but these segments clearly overlap by at least n rounds. More precisely, after time at most $2n(2\lfloor \log l \rfloor + 6)$, there is a segment of n consecutive rounds in which one agent stays and the other moves in one direction. This must result in a rendezvous. Thus we have the following result which should be compared to the lower bound from Proposition 1. (Note that this lower bound holds even when agents know n .)

Theorem 4. *For a ring of known size n , rendezvous can be accomplished in $O(n \log l)$ rounds.*

The above idea cannot be used for rings of unknown size, hence we give a different algorithm working without this additional assumption. We first present the idea of the algorithm. Without loss of generality assume that $L_1 > L_2$. Our goal is to have agent 1 find agent 2 by keeping the latter still for a sufficiently long time, while agent 1 moves along the ring. Since agents do not know whose label is larger, we schedule alternating segments of activity and passivity of increasing length, in such a way that the segments of agent 1 outgrow those of agent 2. The algorithm is formulated for an agent with label L .

Algorithm Ring-Arbitrary-Startup

For $k = 1, 2, \dots$ do

- (1) Move for kL rounds in one (arbitrary) direction;
- (2) Stay for kL rounds.

Theorem 5. *Algorithm Ring-Arbitrary-Startup accomplishes rendezvous in $O(l\tau + \ln^2)$ rounds.*

Proof. Without loss of generality assume that $L_1 > L_2$. First suppose that agent 2 starts before agent 1. Agent 1 performs active and passive segments of length kL_1 from round $k(k-1)L_1 + 1$ to round $k(k+1)L_1$. The length of the

time segment of agent 1, containing round t , is $\lfloor 1/2 + \sqrt{1/4 + (t-1)/L_1} \rfloor L_1$. Similarly, the length of the segment of agent 2, containing round t , is $\lfloor 1/2 + \sqrt{1/4 + (t+\tau-1)/L_2} \rfloor L_2$. There exists a constant c such that after round cn^2 every passive segment of agent 2 is of length greater than n . It now remains to establish when the active segments of agent 1 are sufficiently longer than those of agent 2. When the difference is $2n$ or larger, there are at least n consecutive rounds where agent 1 moves (and thus visits every node of the ring), while agent 2 stays. In the worst case $L_1 = L_2 + 1 = l + 1$ and the inequality $\lfloor 1/2 + \sqrt{1/4 + (t-1)/(l+1)} \rfloor (l+1) - \lfloor 1/2 + \sqrt{1/4 + (t+\tau-1)/l} \rfloor l \geq 2n$ is satisfied by some $t \in O(l\tau + ln^2)$.

If agent 2 starts after agent 1, the condition that the length of the passive segments of agent 2 is of length at least n is still satisfied after round cn^2 , for some constant c , and the second condition (concerning the difference between the agents' segments) is satisfied even sooner than in the first case.

Rendezvous is accomplished by the end of the segment containing round $t \in O(l\tau + ln^2)$. Since the length of this segment is also $O(l\tau + ln^2)$, this concludes the proof. $\square$

In the above upper bound there is a factor l instead of $\log l$ from the simultaneous startup scenario. It remains open if l is a lower bound for rendezvous time in the ring with arbitrary startup.

4 Rendezvous in Arbitrary Connected Graphs

Simultaneous startup. For the scenario with simultaneous startup in arbitrary connected graphs, we will use techniques from Section 3.1, together with the following lemma.

Lemma 3. *Every node within distance D of a node v in a connected graph of maximum degree Δ , can be visited by an agent, starting in v and returning to v , in $O(D\Delta^D)$ rounds.*

Proof. Apply breadth-first search. There are $O(\Delta^D)$ paths of length at most D originating in node v . Thus, in $O(D\Delta^D)$ rounds, all of these paths are explored and all nodes within distance D are visited. $\square$

We keep the exact pattern of activity and passivity from the interleaving algorithm of Section 3.1 but replace the linear walk from the starting node by a breadth-first search walk: if allotted time in a given phase is t , the agent performs breadth-first search for $t/2$ rounds and then backtracks to the starting node. Since the only difference is that we now require a phase of length $O(D\Delta^D)$ to accomplish rendezvous, instead of a phase of length $O(D)$ for the ring, we get the following result.

Theorem 6. *Rendezvous can be accomplished in $O(D\Delta^D \log l)$ rounds in an arbitrary connected graph with simultaneous startup.*

Note that agents do not need to know the maximum degree Δ of the graph to perform the above algorithm. Also note that the above result is optimal for

bounded distance D between agents and bounded maximum degree Δ , since $\Omega(\log l)$ is a lower bound.

Arbitrary startup. We finally show that rendezvous is feasible even in the most general situation: that of an arbitrary connected graph and arbitrary startup. The idea of the algorithm is to let the agent with smaller label be active and the agent with larger label be passive for a sufficiently long sequence of rounds to allow the smaller labeled agent to find the other. This is accomplished, as in the corresponding scenario for the ring, by an increasing sequence of time segments of activity and passivity. However, this time we need much longer sequences of rounds. The algorithm is formulated for an agent with label L .

Algorithm General-Graph-Arbitrary-Startup For $k = 1, 2, \dots$ do

- (1) Perform breadth-first search for $k10^L$ rounds;
- (2) Stay for $k10^L$ rounds.

Theorem 7. *Algorithm General-Graph-Arbitrary-Startup accomplishes rendezvous.*

Proof. Without loss of generality assume that $L_1 > L_2$. First suppose that agent 2 starts before agent 1. There exists a positive integer t such that, after t rounds, we have: (1) the length of (active) segments of agent 2 is $> n^n$, and (2) length of (passive) segments of agent 1 is at least three times larger than the active (and passive) segments of agent 2. Statement 1 is obviously correct, since the lengths of the segments form an increasing sequence of integers. Statement 2 is true, since the ratio of the length of segments of agent 1 and the length of segments of agent 2 is $\sqrt{\frac{10^{L_1+1}t}{10^{L_2}(t+\tau)}} \geq \sqrt{\frac{10t}{(t+\tau)}} \geq 3$, for sufficiently large t . (This is the reason for choosing base 10 for time segments of length $k10^L$). Hence, after t rounds, two complete consecutive segments of agent 2 (one segment active and one segment passive) are contained in a passive segment of agent 1. Since the active segment of agent 2 is of size larger than n^n , this guarantees rendezvous. If agent 2 starts after agent 1, the above conditions are satisfied even sooner. $\square$

Note that the argument to prove correctness of Algorithm Ring-Arbitrary-Startup cannot be directly used for arbitrary connected graphs. Indeed, in the general case, it is not sufficient to show that an arbitrarily large part of an active segment of one agent is included in a passive segment of the other. Instead, since breadth-first search is used, we require a stronger property: the inclusion of an entire active segment (or a fixed fraction of it). This, in turn, seems to require segments of size exponential in L . We do not know if this can be avoided.

5 Conclusion

The rendezvous problem is far from being completely understood even for rings. While for simultaneous startup, we established that optimal rendezvous time is $\Theta(D \log l)$, our upper bound on rendezvous time in rings for arbitrary startup contains a factor l , instead of $\log l$. It remains open if l is also a lower bound in this case. For arbitrary connected graphs we proved feasibility of rendezvous

even with arbitrary startup but our rendezvous algorithm is very inefficient in this general case. The main open problem is to establish if fast rendezvous is possible in the general case. More specifically: question **Q2** from the introduction remains unsolved in its full generality.

Acknowledgements. Andrzej Pelc is supported in part by NSERC grant OGP 0008136 and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais. This work was done during the first and second authors visit at the Research Chair in Distributed Computing of the Université du Québec en Outaouais.

References

1. S. Alpern. The rendezvous search problem. *SIAM J. on Control and Optimization* 33(3), pp. 673–683, 1995.
2. S. Alpern. Rendezvous search on labelled networks. *Naval Research Logistics* 49, pp. 256–274, 2002.
3. S. Alpern and S. Gal. The theory of search games and rendezvous. *Int. Series in Operations research and Management Science*, number 55, Kluwer Academic Publishers, 2002.
4. J. Alpern, V. Baston, and S. Essegiaier. Rendezvous search on a graph. *Journal of Applied Probability* 36(1), pp. 223–231, 1999.
5. S. Alpern and S. Gal. Rendezvous search on the line with distinguishable players. *SIAM J. on Control and Optimization* 33, pp. 1270–1276, 1995.
6. E. Anderson and R. Weber. The rendezvous problem on discrete locations. *Journal of Applied Probability* 28, pp. 839–851, 1990.
7. E. Anderson and S. Essegiaier. Rendezvous search on the line with indistinguishable players. *SIAM J. on Control and Optimization* 33, pp. 1637–1642, 1995.
8. E. Anderson and S. Fekete. Asymmetric rendezvous on the plane. *Proc. 14th Annual ACM Symp. on Computational Geometry*, 1998.
9. E. Anderson and S. Fekete. Two-dimensional rendezvous search. *Operations Research* 49, pp. 107–118, 2001.
10. V. Baston and S. Gal. Rendezvous on the line when the players’ initial distance is given by an unknown probability distribution. *SIAM J. on Control and Optimization* 36, pp. 1880–1889, 1998.
11. V. Baston and S. Gal. Rendezvous search when marks are left at the starting points. *Naval Res. Log.* 48, pp. 722–731, 2001.
12. P. Flocchini, G. Prencipe, N. Santoro, P. Widmayer, Gathering of asynchronous oblivious robots with limited visibility, *Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, LNCS 2010, pp. 247–258, 2001.
13. S. Gal. Rendezvous search on the line. *Operations Research* 47, pp. 974–976, 1999.
14. J. Howard. Rendezvous search on the interval and circle. *Operation research* 47(4), pp. 550–558, 1999.
15. W. Lim and S. Alpern. Minimax rendezvous on the line. *SIAM J. on Control and Optimization* 34(5), pp. 1650–1665, 1996.
16. T. Schelling. *The strategy of conflict*. Oxford University Press, Oxford, 1960.
17. L. Thomas. Finding your kids when they are lost. *Journal on Operational Res. Soc.* 43, pp. 637–639, 1992.

Fast Integer Programming in Fixed Dimension

Friedrich Eisenbrand

Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken,
Germany, eisen@mpi-sb.mpg.de

Abstract. It is shown that the optimum of an integer program in fixed dimension, which is defined by a fixed number of constraints, can be computed with $O(s)$ basic arithmetic operations, where s is the binary encoding length of the input. This improves on the quadratic running time of previous algorithms which are based on Lenstra's algorithm and binary search.

It follows that an integer program in fixed dimension, which is defined by m constraints, each of binary encoding length at most s , can be solved with an expected number of $O(m + \log(m)s)$ arithmetic operations using Clarkson's random sampling algorithm.

1 Introduction

An *integer program* is a problem of the following kind. Given an integral matrix $A \in \mathbb{Z}^{m \times n}$ and integral vectors $b \in \mathbb{Z}^m$, $d \in \mathbb{Z}^n$, determine

$$\max\{d^T x \mid Ax \leq b, x \in \mathbb{Z}^n\}. \quad (1)$$

It is well known [6] that integer programming is NP-complete. The situation changes, if the number of variables or the *dimension* is fixed. For this case, Lenstra [13] showed that (1) can be solved in polynomial time. Lenstra's algorithm does not solve the integer programming problem directly. Instead, it is an algorithm for the *integer feasibility problem*. Here, the task is to find an integer point which satisfies all the constraints, or to assure that $Ax \leq b$ is integer infeasible. If $Ax \leq b$ consists of m constraints, each of binary encoding length $O(s)$, then Lenstra's algorithm requires $O(m + s)$ arithmetic operations on rational numbers of size $O(s)$. The actual integer programming problem (1) can then be solved via *binary search*. It is known [15, p. 239] that, if there exists an optimal solution, then there exists one with binary encoding length $O(s)$. Consequently, the integer programming problem can be solved with $O(ms + s^2)$ arithmetic operations on $O(s)$ -bit numbers. Lenstra's algorithm was subsequently improved [9, 1] by reducing the dependence of the complexity on the dimension n . However, these improvements do not affect the asymptotic complexity of the integer programming problem in fixed dimension. Unless explicitly stated, we from now-on assume that the dimension n is fixed.

Clarkson [2] presented a random sampling algorithm to reduce the dependence of the complexity on the number of constraints.¹ His result is the following. An integer program which is defined by m constraints can be solved with $O(m)$ basic operations and $O(\log m)$ calls to an algorithm which solves an integer program defined by a fixed size subset of the constraints, see also [7].

In light of these results, we are motivated to find a faster algorithm for the integer programming problem in fixed dimension with a fixed number of constraints. It is known [4] that the 2-dimensional integer programming problem with a fixed number of constraints can be solved in linear time. We generalize this to any fixed dimension.

Theorem 1. *An integer program of binary encoding length s in fixed dimension, which is defined by a fixed number of constraints, can be solved with $O(s)$ arithmetic operations on rational numbers of binary encoding length $O(s)$.*

With Clarkson's result, Theorem 1 implies that an integer program which is defined by m constraints, each of binary encoding length $O(s)$ can be solved with an expected number of $O(m + \log(m)s)$ arithmetic operations on rational numbers of binary encoding length $O(s)$. Our result was also motivated by the following fact. The greatest common divisor of two integers can be formulated as an integer program in fixed dimension with a fixed number of constraints, see, e.g., [11]. Our result matches the complexity of the integer programming approach to the gcd with the complexity of the Euclidean algorithm.

Outline of our method. As in Lenstra's algorithm, we make use of the lattice width concept. Let $K \subseteq \mathbb{R}^n$ be a full-dimensional convex body. The *width* of K along a direction $c \in \mathbb{R}^n$ is the quantity $w_c(K) = \max\{c^T x \mid x \in K\} - \min\{c^T x \mid x \in K\}$. The *width* of K , $w(K)$, is the minimum of its widths along nonzero integral vectors $c \in \mathbb{Z}^n \setminus \{0\}$. If K does not include any lattice points, then K must be "flat". This fact is known as Khinchin's *flatness theorem* (see [10]).

Theorem 2 (Flatness theorem). *There exists a constant f_n depending only on the dimension n , such that each full-dimensional convex body $K \subseteq \mathbb{R}^n$, containing no integer points has width at most f_n .*

This fact is exploited in Lenstra's algorithm [13,8] for the integer feasibility problem as follows. If one has to decide, whether a full-dimensional polyhedron P is integer feasible or not, one computes a *flat direction* of P , which is an integral vector $c \in \mathbb{Z}^n \setminus \{0\}$ such that $w(P) \leq w_c(P) \leq \gamma w(P)$ holds for some constant γ depending on the dimension. If $w_c(P)$ is larger than γf_n , then P must contain integer points by the flatness theorem. Otherwise, an integer point of P must lie in one of the constant number of $(n-1)$ -dimensional polyhedra

$$P \cap \{c^T x = \delta\}, \text{ where } \delta \in \mathbb{Z} \cap [\min\{c^T x \mid x \in P\}, \max\{c^T x \mid x \in P\}].$$

¹ Clarkson claims a complexity of $O(m + \log(m)s)$ because he mistakenly relied on algorithms from the literature [13,9,5] for the integer programming problem with a fixed number of constraints, which actually only solve the *integer feasibility problem*.

In this way one can reduce the integer feasibility problem in dimension n to a constant number of integer feasibility problems in dimension $n - 1$.

Our approach is to let the objective function slide into the polyhedron until the width of the truncated polyhedron $P_\pi = P \cap (d^T x \geq \pi)$ is sandwiched between $f_n + 1$ and $\gamma(f_n + 1)$. In this way, we assure that the optimum to the integer programming problem lies in the truncation P_π which is still flat along some integer vector c , thereby reducing the *integer programming problem* over an n -dimensional polyhedron to a constant number of *integer programming problems* over the $(n - 1)$ -dimensional polyhedra

$$P_\pi \cap (c^T x = \delta), \text{ where } \delta \in \mathbb{Z} \cap [\min\{c^T x \mid x \in P_\pi\}, \max\{c^T x \mid x \in P_\pi\}].$$

The problem of determining the correct parameter π is referred to as the *approximate parametric lattice width problem*. The 2-dimensional integer programming algorithm of Eisenbrand and Rote [3] makes already use of this concept. In this paper we generalize this approach to any dimension.

1.1 Notation

A *polyhedron* P is a set of the form $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, for some matrix $A \in \mathbb{R}^{m \times n}$ and some vector $b \in \mathbb{R}^m$. The polyhedron is *rational* if both A and b can be chosen to be rational. If P is bounded, then P is called a *polytope*. The *dimension* of P is the dimension of the affine hull of P . The polyhedron $P \subseteq \mathbb{R}^n$ is *full-dimensional*, if its dimension is n . An inequality $c^T x \leq \delta$ defines a *face* $F = \{x \in P \mid c^T x = \delta\}$ of P , if $\delta \geq \max\{c^T x \mid x \in P\}$. If $F \neq \emptyset$ is a face of dimension 0, then F is called a *vertex* of P . A *simplex* is full-dimensional polytope $\Sigma \subseteq \mathbb{R}^n$ with $n + 1$ vertices. We refer to [14] and [15] for further basics of polyhedral theory.

The *size* of an integer z is the number $\text{size}(z) = 1 + \lceil \log_2(|z| + 1) \rceil$. The size of a rational is the sum of the sizes of its numerator and denominator. Likewise, the size of a matrix $A \in \mathbb{Z}^{m \times n}$ is the number of bits needed to encode A , i.e., $\text{size}(A) = \sum_{i,j} \text{size}(a_{i,j})$, see [15, p. 29]. If a polyhedron P is given as $P(A, b)$, then we denote $\text{size}(A) + \text{size}(b)$ by $\text{size}(P)$. A polytope can be represented by a set of constraints, as well as by the set of its vertices. In this paper we concentrate on polyhedra in fixed dimension with a fixed number of constraints. In this case, if a rational polytope is given by a set of constraints $Ax \leq b$ of size s , then the vertex representation $\text{conv}\{v_1, \dots, v_k\}$ can be computed in constant time and the vertex representation has size $O(s)$. The same holds vice versa.

A rational *lattice* in $\mathbb{R}^n$ is a set of the form $\Lambda = \{Ax \mid x \in \mathbb{Z}^n\}$, where $A \in \mathbb{Q}^{n \times n}$ is a nonsingular matrix. This matrix is a *basis* of Λ and we say that Λ is generated by A and we also write $\Lambda(A)$ to denote a lattice generated by a matrix A . A *shortest vector* of Λ is a nonzero member $0 \neq v \in \Lambda$ of the lattice with minimal euclidean norm $\|v\|$. We denote the length of a shortest vector by $\text{SV}(\Lambda)$.

2 Proof of Theorem 1

Suppose we are given an integer program (1) in fixed dimension with a fixed number of constraints of binary encoding length s . It is very well known that one can assume without loss of generality that the polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is bounded and full-dimensional and that the objective is to find an integer vector with maximal first component. A transformation to such a *standard form problem* can essentially be done with a constant number of Hermite-Normal-Form computations and linear programming. Since the number of constraints is fixed, this can thus be done with $O(s)$ arithmetic operations on rational numbers of size $O(s)$.

Furthermore, we can assume that P is a *two-layer simplex* Σ . A two-layer simplex is a simplex, whose vertices can be partitioned into two sets V and W , such that the first components of the elements in V and W agree, i.e., for all $v_1, v_2 \in V$ one has $v_1(1) = v_2(1)$ and for all $w_1, w_2 \in W$ one has $w_1(1) = w_2(1)$. An integer program over P can be reduced to the disjunction of integer programs over two-layer simplices as follows. First, compute the list of the first components $\alpha_1, \dots, \alpha_\ell$ of the vertices of P in decreasing order. The optimal solution of IP over P is the largest optimal solution of IP over polytopes

$$P_i = P \cap (x(1) \leq \alpha_i) \cap (x(1) \geq \alpha_{i+1}), i = 1, \dots, \ell - 1. \quad (2)$$

Carathéodory's theorem, see [15, p. 94], implies that each P_i is covered by the two-layer simplices, which are spanned by the vertices of P_i . Thus we assume that an integer program has the following form.

Problem 1 (IP). Given an integral matrix $A \in \mathbb{Z}^{n+1 \times n}$ and an integral vector $b \in \mathbb{Z}^{n+1}$ which define a two-layer simplex $\Sigma = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, determine

$$\max\{x(1) \mid x \in P \cap \mathbb{Z}^n\}. \quad (3)$$

The *size* of an IP is the sum of the sizes of A and b .

Our main theorem is proved by induction on the dimension. We know that it holds for $n = 1, 2$ [4,17]. The induction step is by a series of reductions, for which we now give an overview.

- (Step 1) We reduce IP over a two-layer simplex Σ to the problem of determining a parameter π , such that the width of the truncated simplex $\Sigma \cap (x(1) \geq \pi)$ is sandwiched between $f_n + 1$ and $(f_n + 1) \cdot \gamma$, where γ is a constant which depends on the dimension only. This problem is the *approximate parametric lattice width* problem.
- (Step 2) We reduce the approximate parametric lattice width problem to an *approximate parametric shortest vector problem*. Here one is given a lattice basis A and parameters U and k . The task is to find a parameter p such that the length of the shortest vector of the lattice generated $A_{p,k}$ is sandwiched between U and $\gamma' U$, where γ' is a constant which depends on the dimension only. Here $A_{p,k}$ denotes the matrix, which evolves from A by scaling the first k rows with p .

(Step 3) We show that an approximate parametric shortest vector problem can be solved in linear time with a sequence of calls to the LLL-algorithm.

The linear complexity of the parametric shortest vector problem carries over to the integer programming problem with a fixed number of constraints, if we can ensure the following conditions for each reduction step.

(C-1) A problem of size s is reduced to a constant number of problems of size $O(s)$.

(C-2) The size of the rational numbers which are manipulated in the course of the reduction of a problem of size s , do not grow beyond $O(s)$.

At the end of each reduction step, we clarify that the conditions (C-1) and (C-2) are fulfilled.

2.1 Reduction to the Parametric Lattice Width Problem

The *parametric lattice width problem* for a two-layer simplex Σ is defined as follows.

Problem 2 (PLW). Given a two-layer simplex $\Sigma \subseteq \mathbb{R}^n$ and some $K \in \mathbb{N}$, find a parameter π such that the width of the truncated simplex $\Sigma_\pi = \Sigma \cap (x(1) \geq \pi)$ satisfies

$$K \leq w(\Sigma_\pi) \leq 2^{(n+1)/2+2} \cdot \lceil \sqrt{n} \rceil \cdot K, \quad (4)$$

or assert that $w(\Sigma) \leq 2^{(n+1)/2+2} \cdot \lceil \sqrt{n} \rceil \cdot K$.

Let us motivate this concept. Denote the constant $2^{(n+1)/2+2} \cdot \lceil \sqrt{n} \rceil$ by γ . Run an algorithm for PLW on input Σ and $f_n + 1$. If this returns a parameter π such that $f_n + 1 \leq w(\Sigma_\pi) \leq \gamma(f_n + 1)$, then the optimum solution of the IP over Σ must be in the truncated simplex Σ_π . This follows from the fact that we are searching an integer point with maximal first component, and that the truncated polytope has to contain integer points by the flatness theorem. On the other hand, this truncation Σ_π is flat along some integer vector c . Thus the optimum of IP is the largest optimum of the constant number of the $n - 1$ -dimensional integer programs

$$\max\{x(1) \mid x \in (\Sigma_\pi \cap (c^T x = \alpha)) \cap \mathbb{Z}^n\}, \quad (5)$$

where $\alpha \in \mathbb{Z} \cap [\min\{c^T x \mid x \in \Sigma_\pi\}, \max\{c^T x \mid x \in \Sigma_\pi\}]$. This means that we have reduced the integer programming problem over a two-layer simplex in dimension n to a constant number of integer programming problems in dimension $n - 1$ with a fixed number of constraints.

If the algorithm for PLW asserts that $w(\Sigma) \leq \gamma K$, then Σ itself is already flat along an integral direction c . Similarly in this case, the optimization problem can be reduced to a constant number of optimization problems in lower dimension.

Analysis. If the size of Σ and K is at most s and PLW can be solved in $O(s)$ steps with rational numbers of size $O(s)$, then the parameter π which is returned has size $O(s)$. A flat direction of Σ_π can be computed with $O(s)$ arithmetic operations on rationals of size $O(s)$. In fact, a flat direction is a by-product of our algorithm for the approximate parametric shortest vector problem below. It follows that the constant number of $n - 1$ -dimensional IP's (5) have size $O(s)$. These can then be transformed into IP's in standard form with $n - 1$ variables and a constant number of constraints, in $O(s)$ steps. Consequently we have the following lemma.

Lemma 1. *Suppose that PLW for a two-layer simplex Σ and parameter K with $\text{size}(\Sigma) + \text{size}(K) = s$ can be solved with $O(s)$ operations on rational numbers of size $O(s)$, then IP over Σ can also be solved with $O(s)$ operations with rational numbers of size $O(s)$.*

2.2 Reduction to the Approximate Parametric Shortest Vector Problem

In this section we show how to reduce PLW for a two-layer simplex $\Sigma = \text{conv}(V \cup W)$ and parameter K to an *approximate parametric shortest vector* problem. The width of a polyhedron is invariant under translation. Thus we can assume that $0 \in V$ and that the first component of the vertices in W is negative.

Before we formally describe our approach, let us explain the idea with the help of Figure 1. Here we have a two-layer simplex Σ in 3-space. The set V

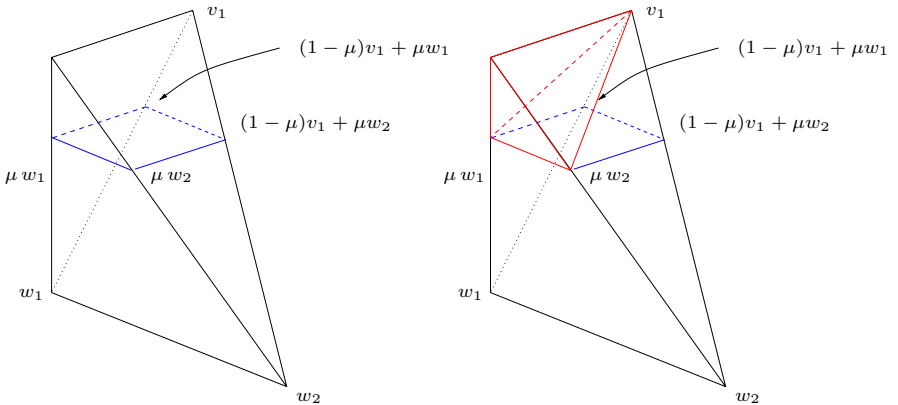


Fig. 1. Solving PLW.

consists of the points 0 and v_1 and W consists of w_1 and w_2 . The picture on the left describes a particular point in time, where the objective function slid into Σ . So we consider the truncation $\Sigma_\pi = \Sigma \cap (x(1) \geq \pi)$ for some $\pi \geq w_1(1)$. This truncation is the convex hull of the points

$$0, v_1, \mu w_1, \mu w_2, (1 - \mu)v_1 + \mu w_1, (1 - \mu)v_1 + \mu w_2, \quad (6)$$

where $\mu = \pi/w_1(1)$. Now consider the simplex $\Sigma_{V,\mu W}$, which is spanned by the points $0, v_1, \mu w_1, \mu w_2$. This simplex is depicted on the right in Figure 1. If this simplex is scaled by 2, then it contains the truncation Σ_π . This is easy to see, since the scaled simplex contains the points $2(1 - \mu)v_1$, $2\mu w_1$ and $2\mu w_2$. So we have the condition $\Sigma_{V,\mu W} \subseteq \Sigma_\pi \subseteq 2\Sigma_{V,\mu W}$. From this we can infer the important observation

$$w(\Sigma_{V,\mu W}) \leq w(\Sigma_\pi) \leq 2w(\Sigma_{V,\mu W}). \quad (7)$$

This means that we can solve PLW for Σ , if we can determine a $\mu \geq 0$, such that the width of the simplex $\Sigma_{V,\mu W}$ is sandwiched between K and $(\gamma/2)K$, where γ denotes the constant $2^{(n+1)/2+2} \cdot \lceil \sqrt{n} \rceil$. We now generalize this observation with the following lemma. A proof is straightforward.

Lemma 2. *Let $\Sigma = \text{conv}(V \cup W) \subseteq \mathbb{R}^n$ be a two-layer simplex, where $0 \in V$, $w(1) < 0$ for all $w \in W$ and let π be a number with $0 \geq \pi \geq w(1)$, $w \in W$. The truncated simplex $\Sigma_\pi = \Sigma \cap (x(1) \geq \pi)$ is contained in the simplex $2\Sigma_{V,\mu W}$, where $\Sigma_{V,\mu W} = \text{conv}(V \cup \mu W)$, where $\mu = \pi/w(1)$, $w \in W$. Furthermore, the following relation holds true*

$$w(\Sigma_{V,\mu W}) \leq w(\Sigma_\pi) \leq 2w(\Sigma_{V,\mu W}). \quad (8)$$

Before we inspect the width of $\Sigma_{V,\mu W}$, let us introduce some notation. We define for an $n \times n$ -matrix A , the matrix $A_{\mu,k}$, as

$$A_{\mu,k}(i, j) = \begin{cases} \mu \cdot A(i, j), & \text{if } i \leq k, \\ A(i, j), & \text{otherwise.} \end{cases} \quad (9)$$

In other words, the matrix $A_{\mu,k}$ results from A by scaling the first k rows with μ .

Suppose that $V = \{0, v_1, \dots, v_{n-k}\}$ and $W = \{w_1, \dots, w_k\}$. Let $A \in \mathbb{R}^{n \times n}$ be the matrix, whose rows are the vectors $w_1^T, \dots, w_k^T, v_1^T, \dots, v_{n-k}^T$ in this order. The width of $\Sigma_{V,\mu W}$ along the vector c can be bounded as

$$\|A_{\mu,k} c\|_\infty \leq w_c(\Sigma_{V,\mu W}) \leq 2\|A_{\mu,k} c\|_\infty, \quad (10)$$

and consequently as

$$(1/\sqrt{n})\|A_{\mu,k} c\| \leq w_c(\Sigma_{V,\mu W}) \leq 2\|A_{\mu,k} c\|. \quad (11)$$

The width of $\Sigma_{V,\mu W}$ is the minimum width along a nonzero vector $c \in \mathbb{Z}^n - \{0\}$. Thus we can solve PLW for a two-layer simplex with parameter K if we can determine a parameter $\mu \in \mathbb{Q}_{>0}$ with

$$\lceil \sqrt{n} \rceil \cdot K \leq \text{SV}(A(A_{\mu,k})) \leq \gamma/4 \cdot K. \quad (12)$$

By substituting $U = \lceil \sqrt{n} \rceil \cdot K$ this reads as follows. Determine a $\mu \in \mathbb{Q}_{>0}$ such that

$$U \leq \text{SV}(\Lambda(A_{\mu,k})) \leq 2^{(n+1)/2} \cdot U. \quad (13)$$

If such a $\mu > 0$ exists, we distinguish two cases. In the first case one has $0 < \mu < 1$. Then $\pi = w(1) \cdot \mu$ is a solution to PLW. In the second case, one has $1 < \mu$ and it follows that $w(\Sigma) \leq \gamma K$. If such a $\mu \in \mathbb{Q}_{>0}$ does not exist, then $\text{SV}(\Lambda(A_{\mu,k})) < U$ for each $\mu > 0$. Also then we assert that $w(\Sigma) \leq \gamma K$.

Thus we can solve PLW for a two-layer simplex $\Sigma = \text{conv}(V \cup W)$ with an algorithm which solves the *approximate parametric shortest vector problem*, which is defined as follows: Given a nonsingular matrix $A \in \mathbb{Q}^{n \times n}$, an integer $1 \leq k \leq n$, and some $U \in \mathbb{N}$, find a parameter $p \in \mathbb{Q}_{>0}$ such that $U \leq \text{SV}(\Lambda(A_{p,k})) \leq 2^{(n+1)/2} \cdot U$ or assert that $\text{SV}(\Lambda(A_{p,k})) \leq 2^{(n+1)/2} \cdot U$ for all $p \in \mathbb{Q}_{>0}$.

We argue now that we can assume that A is an integral matrix and that 1 is a lower bound on the parameter p we are looking for. Clearly we can scale the matrix A and U with the product of the denominators of the components of A . In this way we can already assume that A is integral. If A is integral, then $(|\det(A)|, 0, \dots, 0)$ is an element of $\Lambda(A)$. This implies that we can bound p from below by $1/|\det(A)|$. Thus by scaling U and the last $n - k$ rows of A with $|\det(A)|$, we can assume that $p \geq 1$. Therefore we formulate the approximate parametric shortest vector problem in its integral version.

Problem 3 (PSV). Given a nonsingular matrix $A \in \mathbb{Z}^{n \times n}$, an integer $1 \leq k \leq n$, and some $U \in \mathbb{N}$, find a parameter $p \in \mathbb{Q}_{\geq 1}$ such that $U \leq \text{SV}(\Lambda(A_{p,k})) \leq 2^{(n+1)/2} \cdot U$ or assert that $\text{SV}(\Lambda(A_{p,k})) \leq 2^{(n+1)/2} \cdot U$ for all $p \in \mathbb{Q}_{\geq 1}$ or assert that $\text{SV}(\Lambda(A)) > U$.

By virtue of our reduction to the integral problem, the assertion $\text{SV}(\Lambda(A)) > U$ can never be met in our case. It is only a technicality for the description and analysis of our algorithm below.

Analysis. The conditions (C-1) and (C-2) are straightforward since the binary encoding lengths of the determinant and the products of the denominators are linear in the encoding length of the input in fixed dimension.

Lemma 3. *Suppose that a PSV of size s can be solved with $O(s)$ arithmetic operations on rational numbers of size $O(s)$, then a PLW of size s for a two-layer simplex Σ and parameter K can also be solved with $O(s)$ arithmetic operations on rational numbers of size $O(s)$.*

2.3 Solving the Approximate Parametric Shortest Vector Problem

In the following, we do not treat the dimension as a constant.

The LLL Algorithm

First, we briefly review the *LLL-algorithm* for lattice-basis reduction [12]. We refer the reader to the book of Grötschel, Lovász and Schrijver [8] or von zur Gathen and Gerhard [16] for a more detailed account.

Intuitively, a lattice basis is reduced, if it is “almost orthogonal”. Reduction algorithms apply unimodular transformations of a lattice basis from the right, to obtain a basis whose vectors are more and more orthogonal.

The *Gram-Schmidt orthogonalization* $(b_1^*, \dots, b_n^*)$ of a basis $(b_1, \dots, b_n)$ of $\mathbb{R}^n$ satisfies

$$b_j = \sum_{i=1}^j \mu_{ji} b_i^*, \quad j = 1, \dots, n, \quad (14)$$

where each $\mu_{jj} = 1$. A lattice basis $B \in \mathbb{Z}^{n \times n}$ is *LLL-reduced*, if the following conditions hold for its Gram-Schmidt orthogonalization.

- (i) $|\mu_{i,j}| \leq 1/2$, for every $1 \leq i < j \leq n$;
- (ii) $\|b_{j+1}^* + \mu_{j+1,j} b_j^*\|^2 \geq 3/4 \|b_j^*\|^2$, for $j = 1, \dots, n-1$.

The LLL-algorithm iteratively *normalizes* the basis, which means that the basis is unimodularly transformed into a basis which meets condition (i), and *swaps* two columns if these violate condition (ii). These two steps are repeated until the basis is LLL-reduced. The first column of an LLL-reduced basis is a $2^{(n-1)/2}$ -factor approximation to the shortest vector of the lattice.

Algorithm 1: LLL

Input: Lattice basis $A \in \mathbb{Z}^{n \times n}$.

Output: Lattice basis $B \in \mathbb{Z}^{n \times n}$ with $\Lambda(A) = \Lambda(B)$ and

$$\|b_1\| \leq 2^{(n-1)/2} \text{SV}(\Lambda(A)).$$

- (1) $B \leftarrow A$
- (2) Compute GSO b_j^*, μ_{ji} of B as in equation (14).
- (3) **repeat**
- (4) **foreach** $j = 1, \dots, n$
- (5) **foreach** $i = 1, \dots, j-1$
- (6) $b_j \leftarrow b_j - \lceil \mu_{ji} \rceil b_i$
- (7) **if** There is a subscript j which violates condition (ii)
- (8) Swap columns b_j and b_{j+1} of B
- (9) Update GSO b_j^*, μ_{ji}
- (10) **until** B is LLL-reduced
- (11) **return** B

The key to the termination argument of the LLL-algorithm is the following potential function $\phi(B)$ of a lattice basis $B \in \mathbb{Z}^{n \times n}$:

$$\phi(B) = \|b_1^*\|^{2n} \|b_2^*\|^{2(n-1)} \dots \|b_n^*\|^2. \quad (15)$$

The potential of an integral lattice basis is always an integer. Furthermore, if B_1 and B_2 are two subsequent bases at the end of the repeat-loop of Algorithm 1, then

$$\phi(B_2) \leq \frac{3}{4}\phi(B_1). \quad (16)$$

The potential of the input A can be bounded by $\phi(A) \leq (\|a_1\| \cdots \|a_n\|)^{2n}$. The number of iterations can thus be bounded by $O(n(\log \|a_1\| + \dots + \|a_n\|))$. Step (2) is executed only once and costs $O(n^3)$ operations. The number of operations performed in one iteration of the repeat-loop can be bounded by $O(n^3)$. The rational numbers during the course of the algorithm have polynomial binary encoding length. This implies that the LLL-algorithm has polynomial complexity.

Theorem 3 (Lenstra, Lenstra and Lovász). *Let $A \in \mathbb{Z}^{n \times n}$ be a lattice basis and let A_0 be the number $A_0 = \max\{\|a_j\| \mid j = 1, \dots, n\}$. The LLL-algorithm performs $O(n^4 \log A_0)$ arithmetic operations on rational numbers, whose binary encoding length is $O(n \log A_0)$.*

An Algorithm for PSV

Suppose we want to solve PSV on input $A \in \mathbb{Z}^{n \times n}$, $U \in \mathbb{N}$ and $1 \leq k \leq n$. The following approach is very natural. We use the LLL-algorithm to compute approximate shortest vectors of the lattices $\Lambda(A_{p,k})$ for parameters $p = 2^{\lceil \log U \rceil - i}$ with increasing i , until the approximation of the shortest vector, returned by the LLL-algorithm for $\Lambda(A_{p,k})$ is at most $2^{(n-1)/2} \cdot U$.

Before this is done, we try to assert that $\text{SV}(\Lambda(A_{p,k})) \leq 2^{(n+1)/2} \cdot U$ holds for all $p \in \mathbb{Q}_{\geq 1}$. This is the case if and only if the sub-lattice A' of $\Lambda(A)$, which is defined by $A' = \{v \in \Lambda \mid v(1) = \dots = v(k) = 0\}$ contains already a nonzero vector of at most this length. A basis B' of A' can be read off the Hermite-Normal-Form of A . The first step of the algorithm checks whether the LLL-approximation of the shortest vector of A' has length at most $2^{(n-1)/2} \cdot U$. If this is not the case, then there must be a $p \geq 1$ such that $\text{SV}(\Lambda(A_{p,k})) > U$.

As the algorithm enters the repeat-loop, we can then be sure that the length of the shortest vector of $\Lambda(B)$ is at least U . In the first iteration, this is ensured by the choice of the initial p and the fact that the length of the shortest vector of A' is at least U . In the following iterations, this follows, since the shortest vector of $\Lambda(B)$ has length at least $\|b_1\|/2^{(n-1)/2} > U$. Consider now the iteration where the condition $\|b_1\| \leq 2^{(n-1)/2} \cdot U$ is met. If we scale the first k components of b_1 by 2, we obtain a vector $b' \in \Lambda(A_{2p,k})$. The length of b' satisfies $\|b'\| \leq 2 \cdot \|b_1\| \leq 2^{(n+1)/2} \cdot U$. On the other hand, we argued above that $\text{SV}(\Lambda_{2p,k}) \geq U$. Last, if the condition in step (6) is satisfied, then we can assure that $\text{SV}(\Lambda(A)) > U$. This implies the correctness of the algorithm.

Analysis. Let $B^{(0)}, B^{(1)}, \dots, B^{(s)}$ be the values of B in the course of the algorithm at the beginning of the repeat-loop (step (5)) and consider two consecutive bases $B^{(k)}$ and $B^{(k+1)}$ of this sequence. Step (8) decreases the potential of $B^{(k)}$.

Algorithm 2: Iterated LLL**Input:** Lattice basis $A \in \mathbb{Z}^{n \times n}$, parameters $k, U \in \mathbb{N}$, $1 \leq k \leq n$.

- (1) Compute basis B' of A' , $B' \leftarrow \text{LLL}(B')$
- (2) **if** $\|b'\| \leq 2^{(n-1)/2} \cdot U$
- (3) **return** $\text{SV}(A(A_{p,k})) \leq 2^{(n+1)/2} \cdot U$ for all $p \in \mathbb{Q}_{\geq 1}$
- (4) $p \leftarrow 2^{\lceil \log U \rceil + 1}$, $B \leftarrow A_{p,k}$
- (5) **repeat**
- (6) **if** $p = 1$
- (7) **return** $\text{SV}(A) > U$
- (8) $B \leftarrow B_{1/2,k}$
- (9) $p \leftarrow p/2$
- (10) $B \leftarrow \text{LLL}(B)$
- (11) **until** $\|b_1\| \leq 2^{(n-1)/2} \cdot U$
- (12) **return** $2p$

Thus by (16), we conclude that the number ℓ of iterations performed by the LLL-algorithm in step (10) satisfies

$$\left(\frac{3}{4}\right)^\ell \phi(B^{(k)}) \geq \phi(B^{(k+1)}). \quad (17)$$

From this we conclude that the overall amount of iterations through the repeat-loop of the calls to the LLL-algorithm in step (10) can be bounded by

$$O(\log \phi(B^{(0)})) = O(\log \phi(A_{U,k})). \quad (18)$$

The potential $\phi(A_{U,k})$ can be bounded by $\phi(A_{U,k}) \leq U^{2n^2} (\|a_1\| \cdots \|a_n\|)^{2n}$. As in the analysis of the LLL-algorithm, let A_0 be the number $A_0 = \max\{\|a_j\| \mid j = 1, \dots, n\}$. The overall number of iterations through the repeat-loop of the LLL-algorithm can be bounded by

$$O(n^2(\log U + \log A_0)). \quad (19)$$

Each iteration performs $O(n^3)$ operations. As far as the binary encoding length of the numbers is concerned, we can directly apply Theorem 3 to obtain the next result.

Theorem 4. *Let $A \in \mathbb{Z}^{n \times n}$ be a lattice basis, $U \in \mathbb{N}$ and $1 \leq k \leq n$ be positive integers. Furthermore let $A_0 = \max\{\|a_j\| \mid j = 1, \dots, n\}$. The parametric shortest vector problem for A , U and k can be solved with $O(n^5(\log U + \log A_0))$ basic arithmetic operations with rational numbers of binary encoding length $O(n(\log A_0 + \log U))$.*

This shows that the complexity of *PSV* in fixed dimension n is linear in the input size and operates on rationals whose size is also linear in the input. This concludes the proof of Theorem 1. $\square$

As a consequence, we obtain the following result using Clarkson's [2] random sampling algorithm.

Theorem 5. *An integer program (1) in fixed dimension n , where the objective vector and each of the m constraints of $Ax \leq b$ have binary encoding length at most s , can be solved with an expected amount of $O(m + \log(m)s)$ arithmetic operations on rational numbers of size $O(s)$.*

Acknowledgement. Many thanks are due to Günter Rote and to an ESA-referee for many helpful comments and suggestions.

References

1. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 601–610. ACM Press, 2001.
2. K. L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *Journal of the Association for Computing Machinery*, 42:488–499, 1995.
3. F. Eisenbrand and G. Rote. Fast 2-variable integer programming. In K. Aardal and B. Gerards, editors, *Integer Programming and Combinatorial Optimization, IPCO 2001*, volume 2081 of *LNCS*, pages 78–89. Springer, 2001.
4. S. D. Feit. A fast algorithm for the two-variable integer programming problem. *Journal of the Association for Computing Machinery*, 31(1):99–113, 1984.
5. A. Frank and É. Tardos. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freeman, 1979.
7. B. Gärtner and E. Welzl. Linear programming—randomization and abstract frameworks. In *STACS 96 (Grenoble, 1996)*, volume 1046 of *Lecture Notes in Comput. Sci.*, pages 669–687. Springer, Berlin, 1996.
8. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.
9. R. Kannan. Minkowski’s convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.
10. R. Kannan and L. Lovász. Covering minima and lattice-point-free convex bodies. *Annals of Mathematics*, 128:577–602, 1988.
11. D. Knuth. *The art of computer programming*, volume 2. Addison-Wesley, 1969.
12. A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Annalen*, 261:515 – 534, 1982.
13. H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538 – 548, 1983.
14. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, 1988.
15. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, 1986.
16. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
17. L. Y. Zamanskij and V. D. Cherkasskij. A formula for determining the number of integral points on a straight line and its application. *Ehkon. Mat. Metody*, 20:1132–1138, 1984.

Correlation Clustering – Minimizing Disagreements on Arbitrary Weighted Graphs

Dotan Emanuel and Amos Fiat

Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel

Abstract. We solve several open problems concerning the correlation clustering problem introduced by Bansal, Blum and Chawla [1]. We give an equivalence argument between these problems and the multicut problem. This implies an $O(\log n)$ approximation algorithm for minimizing disagreements on weighted and unweighted graphs. The equivalence also implies that these problems are APX-hard and suggests that improving the upper bound to obtain a constant factor approximation is non trivial. We also briefly discuss some seemingly interesting applications of correlation clustering.

*There is a correlation between the creative and the screwball.
So we must suffer the screwball gladly.*

Kingman Brewster, Jr. (1919–1988) President Yale University
(1963–1977), US Ambassador to Great Britain (1977–1981),
Master of University College, London (1986–1988).

1 Introduction

1.1 Problem Definition

Bansal, Blum and Chawla [1] present the following clustering problem. We are given a **complete** graph on n vertices, where every edge (u, v) is labelled either $\langle + \rangle$ or $\langle - \rangle$ depending on whether u and v have been deemed to be similar or different. The goal is to produce a partition of the vertices (a clustering) that agrees as much as possible with the edge labels. The number of clusters is not an input to the algorithm and will be determined by the algorithm. *I.e.*, we want a clustering that maximizes the number of $\langle + \rangle$ edges within clusters, plus the number of $\langle - \rangle$ edges between clusters (equivalently, minimizes the number of disagreements: the number of $\langle - \rangle$ edges inside clusters plus the number of $\langle + \rangle$ edges between clusters).

Bansal *et al.*, [1], show the problem to be *NP*-hard. They consider the two natural approximation problems:

- Given a complete graph on n vertices with $\langle + \rangle / \langle - \rangle$ labels on the edges, find a clustering that maximizes the number of agreements.

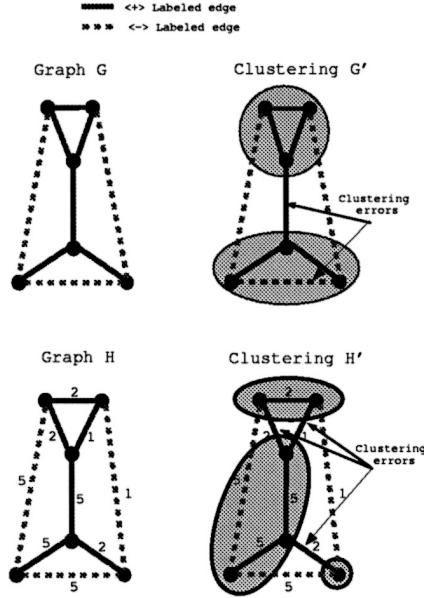


Fig. 1. Two clustering examples for unweighted and weighted (general) graphs. In the unweighted case we give an optimal clustering with two errors: one error on an edge labelled $\langle + \rangle$ and one error on an edge labelled $\langle - \rangle$. For the weighted case we get a different optimal clustering with three errors on $\langle + \rangle$ edges and total weight 5

- Given a complete graph on n vertices with $\langle + \rangle / \langle - \rangle$ labels on the edges, find a clustering that minimizes the number of disagreements.

For the problem of maximizing agreements Bansal *et. al.* ([1]) give a polynomial time approximation scheme. For the problem of minimizing disagreements they give a constant factor approximation. Both of these results hold for complete graphs.

Bansal *et. al.* pose several open problems, including the following:

1. What can one do on general graphs, where not all edges are labelled either $\langle + \rangle$ or $\langle - \rangle$? If $\langle + \rangle$ represents attraction and $\langle - \rangle$ represents the opposite, we may have only partial information on the set of all pairs, or there may be pairs of vertices for which we are indifferent.
2. More generally, for some pairs of vertices, one may be able to quantify the strength of the attraction/rejection. Is it possible to approximate the agreement/disagreement in this case?

In this paper we address these two open questions with respect to minimizing disagreements for unweighted general graphs and for weighted general graphs.

1.2 Problem Variants

Following Bansal *et. al.*, we define three problem variants. For all of these variants, the goal is to find a clustering that maximizes the number of agreements (alternately, minimizes the number of disagreements). In the weighted case one seeks to find a clustering that maximizes the number of agreements weighted by the edge weights (alternately, minimizes the number of disagreements weighted by edge weights).

– Unweighted Complete Graphs

Every pair of vertices has an edge between them, and every edge is labelled either $\langle + \rangle$ or $\langle - \rangle$. An edge labelled $\langle + \rangle$ stands for attraction; the two vertices should be in the same cluster. An edge labelled $\langle - \rangle$ stands for rejection; the two vertices should be in different clusters.

– Unweighted General Graphs

Two vertices need not necessarily have an edge between them, but if so then the edge is labelled either $\langle + \rangle$ or $\langle - \rangle$. If two vertices do not have an edge between them then this represents indifference (or no information) as to whether they should be in the same cluster or not.

– Weighted General Graphs

Two vertices need not necessarily have an edge between them. Edges in the graph have *both* labels $\{\langle + \rangle, \langle - \rangle\}$ and positive real weights. An edge labelled $\langle + \rangle$ with a large weight represents strong attraction (the vertices should be in the same cluster), an edge labelled $\langle - \rangle$ and a large value represents strong rejection (the vertices should not be in the same cluster). No edge, or a weight of zero for an edge represents indifference or no prior knowledge.

For each of these problem variants we focus on minimizing disagreements (as distinct from the easier goal of maximizing agreements). We seek to minimize the number of edges labelled $\langle - \rangle$ within the clusters plus the number of the edges labelled $\langle + \rangle$ that cross cluster boundaries. In the weighted version we seek to minimize the sum of the weights of edges labelled $\langle - \rangle$ within the clusters plus the sum of the weights of the edges labelled $\langle + \rangle$ that cross cluster boundaries.

In the rest of this paper when we refer to the “correlation clustering problem” or “the clustering problem” we mean the problem of minimizing disagreements in one of the problem variants above. We will also say “positive edge” when referring to an edge labelled $\langle + \rangle$ and “negative edge” when referring to an edge labelled $\langle - \rangle$. Note that for both positive and negative edges, the weights are always ≥ 0 .

Remarks:

1. We remark that although the optimal solution to maximizing agreements is the same as the optimal solution to minimizing disagreements, in terms of approximation ratios these two goals are obviously distinct.
2. It is not hard to see that for all problem variants, a trivial algorithm for maximizing agreements gives a factor of two approximation. Simply consider one of the two clusterings: every vertex is a distinct cluster or all vertices are in the same cluster.

3. It should be obvious that the problem of minimizing disagreements for unweighted complete graphs is a special case of minimizing disagreements for unweighted general graphs, which is itself a special case of minimizing disagreements for weighted general graphs.
4. We distinguish between the different problems because the approximation results and the hardness of approximation results are different or mean different things in the different variants.

1.3 Our Contributions

In [1] the authors presented a constant factor approximation algorithm for the problem of unweighted complete graphs, and proved that the problem for the weighted general graphs is APX-Hard. They gave the problem of finding approximation algorithms and hardness of approximation results for the two other variants (unweighted and weighted general graphs) as open questions.

Problem class	Approximation	Hardness of Approximation	Equivalence
Unweighted complete graphs	$c \in O(1)$	Open	
Unweighted general graphs	Open	Open	
Weighted general graphs	Open	APX-hard	

Fig. 2. Previous Results [BBC 2002] — Minimizing Disagreements

Problem class	Approximation	Hardness of Approximation	Equivalence
Unweighted general graphs	$O(\log n)$	APX-hard	Unweighted multicut
Weighted general graphs	$O(\log n)$		Weighted multicut

Fig. 3. Our Contributions. The equivalence column is to say that any c -approximation algorithm for one problem will translate into a c' -approximation approximation for the other, where c and c' are constants.

We give an $O(\log n)$ approximation algorithm for minimizing disagreements for both the weighted and unweighted general graph problems, and prove that the problem is APX-hard even for the unweighted general graph problem, thus admitting no polynomial time approximation scheme (PTAS). We do this by reducing the correlation clustering problems to the multicut problem.

We further show that the correlation clustering problem and the multicut problem are equivalent for both weighted and unweighted versions, and that any constant approximation algorithm or hardness of approximation result for one problem implies the same for the other. Note that the question of whether there exists a constant factor approximation for general weighted and unweighted graphs remains open. This is not very surprising as the multicut problem has been studied at length, and no better approximation found, this suggests that the problem is not trivial.

1.4 Some Background Regarding the Multicut Problem

The weighted multicut problem is the following problem: Given an undirected graph G , a weight function w on the edges of G , and a collection of k pairs of distinct vertices (s_i, t_i) of G , find a minimum weight set of edges of G whose removal disconnects every s_i from the corresponding t_i .

The problem was first stated by Hu in 1963 [8]. For $k = 1$, the problem coincides of course with the ordinary min cut problem. For $k = 2$, it can be also solved in polynomial time by two applications of a max flow algorithm [16]. The problem was proven NP-hard and MAX SNP-hard for any $k \geq 3$ in by Dahlhaus, Johnson, Papadimitriou, Seymour and Yannakakis [5]. The best known approximation ratio for weighted multicut in general graphs is $O(\log k)$ [7]. For planar graphs, Tardos and Vazirani [13] give an approximate Max-Flow Min-Cut theorem and an algorithm with a constant approximation ratio. For trees, Garg, Vazirani and Yannakakis give an algorithm with an approximation ratio of two [6].

1.5 Structure of This Paper

In section 2 we give notations and definitions, in section 3 we prove approximation results, and in section 4 we establish the equivalence of the multicut and correlation clustering problems. Section 5 gives the APX-hardness proofs.

2 Preliminaries

Let $G = (V, E)$ be a graph on n vertices. Let $e(u, v)$ denote the label $(\langle + \rangle, \langle - \rangle)$ of the edge (u, v) . Let $E^{\langle + \rangle}$ be the set of positive edges and let $G^{\langle + \rangle}$ be the graph induced by $E^{\langle + \rangle}$, $E^{\langle + \rangle} = \{(u, v) | e(u, v) = \langle + \rangle\}$, $G^{\langle + \rangle} = (V, E^{\langle + \rangle})$. Let $E^{\langle - \rangle}$ be the set of negative edges and $G^{\langle - \rangle}$ the graph induced by $E^{\langle - \rangle}$, $E^{\langle - \rangle} = \{(u, v) | e(u, v) = \langle - \rangle\}$, $G^{\langle - \rangle} = (V, E^{\langle - \rangle})$.

Definition 2.01 *We will call a cycle $(v_1, v_2, v_3, \dots, v_k)$ in G a **erroneous cycle** if it is a simple cycle, and it contains exactly one negative edge.*

We let OPT denote the optimal clustering on G . In general, for a clustering $\mathcal{C}$, let $C(v)$ be the set of vertices in the same cluster as v . We call an edge (u, v) a positive mistake if $e(u, v) = \langle + \rangle$ and yet $u \notin C(v)$. We call an edge (u, v)

a negative mistake if $e(u, v) = \langle - \rangle$ and $u \in C(v)$. The number of mistakes of a clustering $\mathcal{C}$ is the sum of positive and negative mistakes. The weight of the clustering is the sum of the weights of mistaken edges in $\mathcal{C}$;

$$w(\mathcal{C}) = \sum_{e(u,v)=\langle - \rangle, u \in C(v)} w(u, v) + \sum_{e(u,v)=\langle + \rangle, u \notin C(v)} w(u, v).$$

For a general set of edges $T \subset E$ we will define the weight of T to be the sum of the weights in T , $w(T) = \sum_{e \in T} w(e)$.

For a graph $G = (V, E)$ and a set of edges $T \subset E$ we define the graph $G \setminus T$ to be the graph $(V, E \setminus T)$.

Definition 2.02 *We will call a clustering a **consistent clustering** if it contains no mistakes.*

3 A Logarithmic Approximation Factor for Minimizing Disagreements

3.1 Overview

We now show that finding an optimal clustering is equivalent to finding a minimal weight covering of the erroneous cycles. An edge is said to cover a cycle if the edge disconnects the cycle.

Guided by this observation will define a multicut problem derived from our original graph by replacing the negative edges with source-sink pairs (and some other required changes). We show that a solution to the newly formed multicut problem induces a solution to the clustering problem, that this solution and the multicut solution have the same weight, and that optimal solution to the multicut problem induces an optimal solution to the clustering problem.

These reductions imply that the $O(\log k)$ approximation algorithm for the multicut problem [7] induces an $O(\log n)$ approximation algorithm for the correlation clustering problem. We prove this for weighted general graphs, which imply the same result for unweighted general graphs. We start by stating two simple lemmata:

Lemma 3.11 *A graph contains no erroneous cycles if and only if it has a consistent clustering.*

Proof. Omitted.

Lemma 3.12 *The weight of mistakes made by the optimal clustering is equal to the minimal weight set of edges whose removal will eliminate all erroneous cycles in G .*

Proof. Omitted.

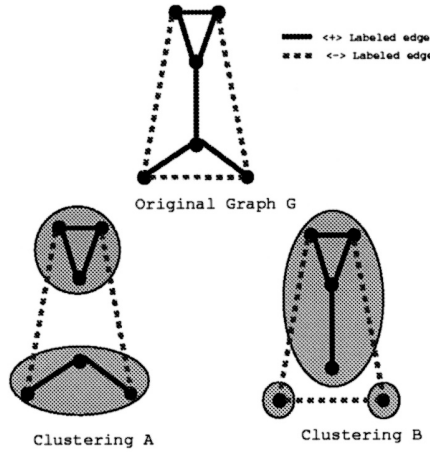


Fig. 4. Two optimal clusterings for G . For both of these clusterings we have removed two edges (different edges) so as to eliminate all the erroneous cycles in G . After the edges were removed every connected component of $G^{(+)}$ is a cluster. Note that the two clusterings are consistent; no positive edges connect two clusters and no negative edges connect vertices within the same cluster.

3.2 Reduction from Correlation Clustering to Weighted Multicut

We give a reduction from the problem of correlation clustering to the weighted multicut problem. The reduction translates an instance of unweighted correlation clustering into an instance of unweighted graph multicut, and an instance of weighted correlation clustering into an instance of weighted graph multicut.

Given a weighted graph G whose edges are labelled $\{ \langle + \rangle, \langle - \rangle \}$ we construct a new graph H_G and a collection of source-sink pairs $S_G = \{ \langle s_i, t_i \rangle \}$ as follows:

- For every negative edge $(u, v) \in E^{(-)}$ we introduce a new vertex $v_{\widehat{u,v}}$, a new edge $(v_{\widehat{u,v}}, u)$ with weight equal to that of (u, v) , and a source-sink pair $\langle v_{\widehat{u,v}}, v \rangle$.
- Let V_{new} denote the set of new vertices, E_{new} the set of new edges, and S_G the set of source-sink pairs. Let $V' = V \cup V_{\text{new}}$, $E' = E^{(+)} \cup E_{\text{new}}$, $H_G = (V', E')$. The weight of the edges in $E^{(+)}$ remains unchanged. We now have a multicut problem on (H_G, S_G) .

We claim that given any solution to the multicut problem, this implies a solution to the correlation clustering problem with the exact same value, and that an approximate solution to the former gives an approximate solution to the later.

Theorem 3.21 *(H_G, S_G) has a cut of weight W if and only if G has a clustering of weight W , and we can easily construct one from the other. In particular, the optimal clustering in G of weight W implies an optimal multicut in (H_G, S_G) of weight W and vice versa.*

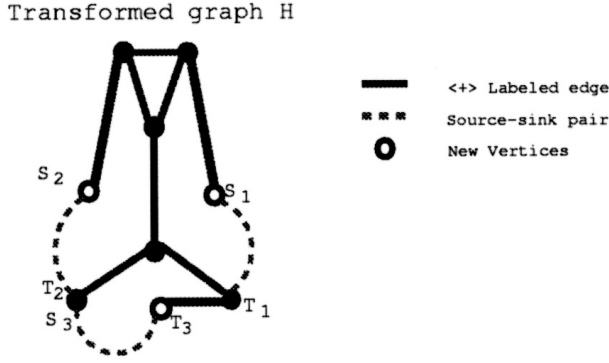


Fig. 5. The original graph from Figure 4 after the transformation

Proof. **Proposition 3.22** Let $\mathcal{C}$ be a clustering on G with weight W then there exists a multicut T' in (H_G, S_G) with weight W .

Proof. Let $\mathcal{C}$ be a clustering of G with weight W , where T is the set of mistakes made by $\mathcal{C}$ ($w(T) = W$). Let $T' = \{(u, v) | (u, v) \in T, (u, v) \in G^{(+)}\} \cup \{(v_{u,v}, u) | (u, v) \in T, (u, v) \in G^{(-)}\}$, i.e., we replace every negative edge $(u, v) \in T$, with the edge $(v_{u,v}, u)$. Note that $w(T) = w(T')$. We now argue that T' is a multicut.

Assume not, then there exists a pair $(v_{u,v}, v) \in S_G$ and a path from $v_{u,v}$ to u that contains no edge from T' . From the construction of S_G and H_G , this implies that the edge $(v_{u,v}, u) \notin T'$ and that there exists a path from u to v in $G^{(+)} \setminus T$. Note that (u, v) is a negative edge in $G \setminus T$, so the negative edge (u, v) and the path from u to v in $G^{(+)} \setminus T$ jointly form an erroneous cycle in $G \setminus T$. This is a contradiction since $G \setminus T$ is consistent (Lemma 3.12) and contains no erroneous cycles (Lemma 3.11). Note that the proof is constructive.

Proposition 3.23 If T' is a multicut in H_G of weight W , then there exists a clustering $\mathcal{C}$ in G of weight W .

Proof. We construct a set T from the cut T' by replacing all edges in E_{new} with the corresponding negative edges in G , and define a clustering $\mathcal{C}$ by taking every connected component of $G^{(+)} \setminus T$ as a cluster. T has the same cardinality and total weight T' . Thus, if we show that $\mathcal{C}$ is consistent on $G \setminus T$ we are done (since $w(\mathcal{C}(G)) = w(\mathcal{C}(G \setminus T)) + w(T) = 0 + w(T') = W$).

Assume that $\mathcal{C}$ is not a consistent clustering on $G \setminus T$, then there exists an erroneous cycle in $G \setminus T$ (Lemma 3.11). Let (u, v) be the negative edge along this cycle. This implies a path from u to v in H_G (the path of positive edges of the cycle in $G \setminus T$). We also know that (u, v) is negative edge, which means that in the construction of H_G we replaced it with edge $(v_{u,v}, u)$. The edge $(v_{u,v}, u)$

is not in the cut (not in T') since (u, v) is not in T (as $(u, v) \in G \setminus T$). From this it follows that there is a path from $v_{u,v}$ to v in H_G . But the pair $\langle v_{u,v}, v \rangle$ are a source-sink pair which is in contradiction to T' being a multicut.

Proposition 3.22 and proposition 3.23 imply that

$$\begin{aligned} w(\text{Optimal clustering}(G)) &= w(\text{Multicut induced by opt. clustering}(H_G, S_G)) \\ &\geq w(\text{Minimal Multicut}(H_G, S_G)) \\ &= w(\text{Clustering on } G \text{ induced by minimal multicut}) \\ &\geq w(\text{Optimal clustering}(G)), \end{aligned}$$

where all inequalities must hold with equalities.

We can now use the approximation algorithm of [7] to get an $O(\log k)$ approximation solution to the multicut problem (k is the number of source-sink pairs) which translates into an $O(\log |E^{(-)}|) \leq O(\log n^2) = O(\log n)$ solution to the clustering problem. Note that this result holds for both weighted and unweighted graphs and that the reduction of the unweighted correlation clustering problem results in a multicut problem with unity capacities and demands.

4 Reduction from Multicut to Correlation Clustering

In the previous section we argued that every correlation clustering problem can be presented (and approximately solved) as a multicut problem. We will now show that the opposite is true as well, that every instance of the multicut problem can be transformed to an instance of a correlation clustering problem, and that transformation has the following properties: any solution to the correlation clustering problem induces a solution to the multicut problem with lower or equal weight, and an optimal solution to the correlation clustering problem induces an optimal solution to the multicut problem.

In the previous section we could use one reduction for the weighted version and the unweighted version. Here we will present two slightly different reductions from unweighted multicut to unweighted correlation clustering and from weighted multicut to weighted correlation clustering.

4.1 Reduction from Weighted Multicut to Weighted Correlation Clustering

Given a multicut problem instance: an undirected graph H , a weight function w on the edges of H , $w : E \rightarrow \mathcal{R}^+$, and a collection of k pairs of distinct vertices $S = \{\langle s_i, t_i \rangle, \dots, \langle s_k, t_k \rangle\}$ of H we construct a correlation clustering problem as follows:

- We start with $G_H = H$, all edge weights are preserved and all edges labelled $\langle + \rangle$.

- In addition, for every source-sink pair $\langle s_i, t_i \rangle$ we add to G_H a negative edge $e_i = (s_i, t_i)$ with weight $w(e_i) = \sum_{e \in H} w(e) + 1$.

Our transformation is polynomial, adds at most $O(n^2)$ edges, and increases the largest weight in the graph by a multiplicative factor of at most n .

Theorem 4.11 *A clustering on G_H with weight W induces a multicut on (H, S) with weight $\leq W$. An optimal clustering in G_H induces an optimal multicut in (H, S) .*

Proof. If a clustering $\mathcal{C}$ on G_H contains no negative mistakes, then the set of positive mistakes T is a multicut on H and $w(c) = w(t)$. If $\mathcal{C}$ contains a negative mistake, say (u, v) , we take one of the endpoints (u or v) and place it in a cluster of it's own, thus eliminating this mistake. Since every negative edge has weight $\geq$ the sum of all positive edges, the gain by splitting the cluster will exceed the loss introduced by new positive mistakes, therefore the new clustering $\mathcal{C}'$ on G has weight $W' < W$, and it contains no negative mistakes. Thus, we know that $\mathcal{C}$ induces a cut of weight W' .

Now let T denote the minimal multicut in (H, S) . T induces a clustering on G_H (the connected components of $G^{(+)} \setminus T$) that contains no negative mistakes. This in turn means that the weight of the clustering is the weight of the positive mistakes, which is exactly $w(T)$. We now have $w(\text{Optimal multicut}) = w(\text{Clustering induced by optimal multicut})$. Combining the above two arguments we have that

$$\begin{aligned} w(\text{Optimal multicut}) &= w(\text{Clustering induced by optimal multicut}) \\ &\geq w(\text{Optimal clustering}) \\ &\geq w(\text{Multicut induced by the optimal clustering}) \\ &\geq w(\text{Optimal multicut}). \end{aligned}$$

Thus, all inequalities must hold with equality.

4.2 Reduction from Unweighted Multicut to Unweighted Correlation Clustering

Given an unweighted multicut problem instance: an undirected graph H and a collection of k pairs of distinct vertices $S = \{\langle s_i, t_i \rangle, \dots, \langle s_k, t_k \rangle\}$ of H we construct an unweighted correlation clustering problem as follows:

- For every v , $\langle v, u \rangle \in S$ or $\langle u, v \rangle \in S$, (v is either a source or a sink) we add $n - 1$ new vertices and connect those vertices and v in a clique with positive edges (weight 1). We denote this clique by Q_v .
- For every pair $\langle s_i, t_i \rangle \in S$ we connect all vertices of Q_{s_i} to t_i and all vertices of Q_{t_i} to s_i using edges labelled $\langle - \rangle$.
- Other vertices of H are added to the vertex set of G_H , Edges of H are added to the edge set of G_H and labelled $\langle + \rangle$.

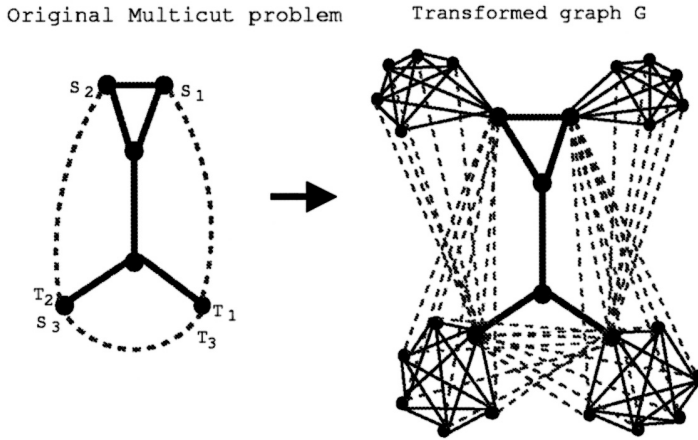


Fig. 6. Transformation from the unit capacity multicut problem (on the left) to the unweighted correlation clustering problem (on the right)

Our goal is to emulate the previous argument for weighted general graphs in the context of unweighted graphs. We do so by replacing the single edge of high weight with many unweighted negative edges. Our transformation is polynomial time, adds at most n^2 vertices and at most n^3 edges.

Theorem 4.21 *A clustering on G_H with weight W induces a multicut on (H, S) with weight $\leq W$. An optimal clustering in G of weight W induces an optimal multicut for (H, S) of weight W .*

Proof. We call a clustering *pure* if all vertices that belong to the same Q_v are in the same cluster, and that if $\langle v, w \rangle \in S$ then Q_v and Q_w are in different clusters. The following proposition implies that we can “fix” any clustering to be a pure clustering without increasing its weight.

Proposition 4.22 *Given a clustering C on G . We can “fix” that clustering to be pure thus find a pure clustering C' on G such that $w(C') \leq w(C)$.*

Proof. For every Q_v that is split amongst two or more cluster we take all vertices of Q_v to form a new cluster. By doing so we may be adding up to $n - 1$ new mistakes, (positive mistakes, positive edges adjacent to v in original graph). Merging these vertices into one cluster component will reduce the number of errors by $n - 1$ at least.

If two Q_v and Q_w are in the same cluster component, we can move one of them into a cluster of its own. As before, we may be introducing as many as $n - 1$ new positive mistakes but simultaneously eliminating $2n$ negative mistakes.

Given a clustering C on G_H we first “fix” it using the technique of proposition 4.22 to obtain a pure clustering C' . Any mistake for pure clustering must be a positive mistake, the only negative edges are between clusters.

Let T be the set of positive mistakes for C' , we now show that T is a multicut on (H, S) . No source-sink pair are in the same cluster since the clustering is pure and removing the edges of T disconnects every source/sink pair. Thus, T is a multicut for (H, S) .

Let OPT be the optimal clustering on G . OPT is pure (otherwise we can fix it and get a better clustering) and therefore induces a multicut on (H, S) . Let T denote the minimal multicut in (H, S) . T induces a pure-clustering on G as follows: take the connected component of $G^{(+)} \setminus T$ as clusters and for every terminal $v \in S$ add every node in Q_v to the cluster containing vertices v . It can be easily seen that this gives a pure clustering, and that the only mistakes on the clustering are the edges in T .

Thus, we can summarize:

$$\begin{aligned} w(\text{Optimal multicut}) &= w(\text{Clustering induced by optimal multicut}) \\ &\geq w(\text{Optimal clustering}) \\ &\geq w(\text{Multicut induced by optimal clustering}) \\ &\geq w(\text{Optimal multicut}). \end{aligned}$$

All inequalities must hold with equality.

5 More on Correlation Clustering and Multicuts

The two way reduction we just presented proves that the correlation clustering problem and the multicut problem are essentially identical problems. Every exact solution to one implies an exact solution to the other. Every polynomial time approximation algorithm with a constant, logarithmic, or polylogarithmic approximation factor for either problem translates into a polynomial time approximation algorithm with a constant, logarithmic or polylogarithmic approximation factor, respectively, for the other. (We use this to prove an $O(\log n)$ approximation in section 3).

From this it also follows that hardness of approximation results transfer from one problem to the other. Since the multicut problem is APX-hard and remains APX-hard even in the unweighted case it implies that unweighted correlation clustering problem is itself APX hard.

An interesting observation is that [1] give a constant factor approximation for the unweighted complete graph. This implies that the unweighted multicut problem where every two nodes u, v , are either connected by an edge or $\langle u, v \rangle$ is a source/sink pair has a constant factor approximation.

On the other hand, correlation clustering problems where $G^{(+)}$ is a planar graph or has a tree structure has a constant factor approximation (as follows from [13,6]).

Addendum: We recently learned that two other groups, Erik D. Demaine and Nicole Immorlica [3] and Charikar, Guruswami, and Wirth [12], have both independently obtained similar results (using somewhat different techniques).

References

1. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Foundations of Computer Science (FOCS)*, pages 238–247, 2002.
2. Grigori Calinescu, Cristina G. Fernandes, and Bruce Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. *proceedings of the 6th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 1998.
3. Demaine Erik D. and Immorlica Nicole. Correlation clustering with partial information. *APPROX*, 2003.
4. E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiway cuts. *Proceedings, 24th ACM Symposium on Theory of Computing*, pages 241–251, 1992.
5. E. Dahlhaus, D.S. Johnson, C.H. Papadimitriou, P.D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 4(23):864–894, 1994.
6. N. Garg, V. Vazirani, and M. Yannakakis. Primal–Dual Approximation Algorithms for Integral Flow and Multicut in Trees, with Applications to Matching and Set Cover. *Proceedings of ICLP*, pages 64–75, 1993.
7. Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. 25th *STOC.*, pages 698–707, 1993.
8. T.C. Hu. Multicommodity network flows. *Operations Research*, (11):344–360, 1963.
9. D. Klein, S. D. Kamvar, and C. D Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. *Proceedings of the Nineteenth International Conference on Machine Learning*, 2002.
10. Tom Leighton and S. Rao. An approximate max-flow mincut theorem for uniform multicommodity flow problems with applications to approximation algorithms. *In Proc. of the 29th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 422–431, 1988.
11. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Symposium on Math, Statistics, and Probability*, pages 281–297, 1967.
12. Venkat Guruswami Moses Charikar and Tony Wirth. Personal communication. 2003.
13. E. Tardos and V. V. Vazirani. Improved bounds for the max flow min multicut ratio for planar and $K_{r,r}$ -free graphs. *Information Processing Letters*, pages 698–707, 1993.
14. K. Wagstaff and C. Cardie. Clustering with instance-level constraints. *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1103–1110, 2000.
15. K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 577–584, 2001.
16. M. Yannakakis, P. C. Kanellakis, S. C. Cosmadakis, and C. H. Papadimitriou. Cutting and partitioning a graph after a fixed pattern. *Proceedings, 10th Intl. Coll. on Automata, Languages and Programming*, page 712–722, 1983.

Dominating Sets and Local Treewidth^{*}

Fedor V. Fomin¹ and Dimitrios M. Thilikos²

¹ Department of Informatics, University of Bergen, N-5020 Bergen, Norway,
fomin@ii.uib.no

² Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de
Catalunya, Campus Nord – Mòdul C5, c/Jordi Girona Salgado 1-3, E-08034,
Barcelona, Spain, sedthilk@lsi.upc.es

Abstract. It is known that the treewidth of a planar graph with a dominating set of size d is $O(\sqrt{d})$ and this fact is used as the basis for several fixed parameter algorithms on planar graphs. An interesting question motivating our study is if similar bounds can be obtained for larger minor closed graph families. We say that a graph family $\mathcal{F}$ has the domination-treewidth property if there is some function $f(d)$ such that every graph $G \in \mathcal{F}$ with dominating set of size $\leq d$ has treewidth $\leq f(d)$. We show that a minor-closed graph family $\mathcal{F}$ has the domination-treewidth property if and only if $\mathcal{F}$ has bounded local treewidth. This result has important algorithmic consequences.

1 Introduction

The last ten years has witnessed the of rapid development of a new branch of computational complexity: parameterized complexity (see the book of Downey & Fellows [9]). Roughly speaking, a parameterized problem with parameter k is *fixed parameter tractable* if it admits an algorithm with running time $f(k)|I|^\beta$. (Here f is a function depending only on k , $|I|$ is the length of the non parameterized part of the input and β is a constant.) Typically, $f(k) = c^k$ is an exponential function for some constant c .

A d -dominating set D of a graph G is a set of d vertices such that every vertex outside D is adjacent to a vertex of D . Fixed parameter version of the dominating set problem (the task is to compute, given a G and a positive integer d , a d -dominating set or to report that no such set exists) is one of the core problems in the Downey & Fellows theory. Dominating set is $W[2]$ complete and thus widely believed to be not fixed parameter tractable. However for planar graphs the situation is different and during the last five years a lot of work was done on fixed parameter algorithms for the dominating set problem on planar graphs and different generalizations of planar graphs. For planar graphs Downey and Fellows [9] suggested an algorithm with running time $O(11^d n)$. Later the running time was reduced to $O(8^d n)$ [2]. An algorithm with a sublinear exponent

^{*} The second author was supported by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity - Future Technologies and by the Spanish CICYT project TIC-2002-04498-C05-03 (TRACER).

for the problem with running time $O(4^{6\sqrt{34d}}n)$ was given by Alber et al. [1]. Recently, Kanj & Perković [16] improved the running time to $O(2^{27\sqrt{d}}n)$ and Fomin & Thilikos to $O(2^{15.13\sqrt{d}}d + n^3 + d^4)$ [13]. The fixed parameter algorithms for extensions of planar graphs like bounded genus graphs and graphs excluding single-crossing graphs as minors are introduced in [10,6].

The main technique to handle the dominating set problem which was exploited in several papers is that every graph G from a given graph family $\mathcal{F}$ with a domination set of size d has treewidth at most $f(d)$, where f is some function depending only on $\mathcal{F}$. With some work (sometimes very technical) a tree decomposition of width $O(f(d))$ is constructed and standard dynamic programming techniques on graphs of bounded treewidth are implemented. Of course this method can not be used for all graphs. For example, a complete graph K_n on n vertices has dominating set of size one and the treewidth of K_n is $n - 1$. So the interesting question here is: Can this 'bounding treewidth method' be extended for larger minor-closed graph classes and what are the restrictions of these extensions?

In this paper we give a *complete* characterization of minor-closed graph families for which the 'bounding treewidth method' can be applied. More precisely, a minor-closed family $\mathcal{F}$ of graphs has the *domination-treewidth property* if there is some function $f(k)$ such that every graph $G \in \mathcal{F}$ with dominating set of size $\leq k$ has treewidth $\leq f(k)$. We prove that any minor-closed graph class has the domination-treewidth property if and only if it is of bounded local treewidth. Our proof is constructive and can be used for constructing fixed parameter algorithms for dominating set on minor-closed families of bounded local treewidth. The proof is based on Eppstein's characterization of minor-closed families of bounded local treewidth [11] and on a modification of the Robertson & Seymour excluded grid minor theorem due to Diestel et al.[8].

2 Definitions and Preliminary Results

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. We let n denote the number of vertices of a graph when it is clear from context. For every nonempty $W \subseteq V(G)$, the subgraph of G induced by W is denoted by $G[W]$. We define the *r-neighborhood* of a vertex $v \in V(G)$, denoted by $N_G^r[v]$, to be the set of vertices of G at distance at most r from v . Notice that $v \in N_G^r[v]$. We put $N_G[v] = N_G^1[v]$. We also often say that a vertex v *dominates* subset $S \subset V(G)$ if $N_G[v] \supseteq S$.

Given an edge $e = \{x, y\}$ of a graph G , the graph G/e is obtained from G by contracting the edge e ; that is, to get G/e we identify the vertices x and y and remove all loops and duplicate edges. A graph H obtained by a sequence of edge contractions is said to be a *contraction* of G . A graph H is a *minor* of a graph G if H is the subgraph of a contraction of G . We use the notation $H \preceq G$ (resp. $H \preceq_c G$) for H a minor (a contraction) of G .

The $m \times m$ grid is the graph on $\{1, 2, \dots, m^2\}$ vertices $\{(i, j) : 1 \leq i, j \leq m\}$ with the edge set

$$\{(i, j)(i', j') : |i - i'| + |j - j'| = 1\}.$$

For $i \in \{1, 2, \dots, m\}$ the vertex set (i, j) , $j \in \{1, 2, \dots, m\}$, is referred as the i th row and the vertex set (j, i) , $j \in \{1, 2, \dots, m\}$, is referred to as the i th column of the $m \times m$ grid.

The notion of treewidth was introduced by Robertson and Seymour [17]. A *tree decomposition* of a graph G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, with $\{X_i \mid i \in I\}$ a family of subsets of $V(G)$ and T a tree, such that

- $\bigcup_{i \in I} X_i = V(G)$.
- For all $\{v, w\} \in E(G)$, there is an $i \in I$ with $v, w \in X_i$.
- For all $i_0, i_1, i_2 \in I$: if i_1 is on the path from i_0 to i_2 in T , then $X_{i_0} \cap X_{i_2} \subseteq X_{i_1}$.

The *width* of the tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth $\mathbf{tw}(G)$ of a graph G is the minimum width of a tree decomposition of G .

We need the following facts about treewidth. The first fact is trivial.

- For any complete graph K_n on n vertices, $\mathbf{tw}(K_n) = n - 1$, and for any complete bipartite graph $K_{n,n}$, $\mathbf{tw}(K_{n,n}) = n$.

The second fact is well known but its proof is not trivial. (See e.g., [7].)

- The treewidth of the $m \times m$ grid is m .

A family of graphs $\mathcal{F}$ is minor-closed if $G \in \mathcal{F}$ implies that every minor of G is in $\mathcal{F}$. Graphs with the domination-treewidth property are the main issue of this paper. We say that a minor-closed family $\mathcal{F}$ of graphs has the *domination-treewidth property* if there is some function $f(d)$ such that every graph $G \in \mathcal{F}$ with dominating set of size $\leq d$ has treewidth $\leq f(d)$.

The next fact we need is the improved version of the Robertson & Seymour theorem on excluded grid minors [18] due to Diestel et al. [8]. (See also the textbook [7].)

Theorem 1 ([8]). *Let r, m be integers, and let G be a graph of treewidth at least $m^{4r^2(m+2)}$. Then G contains either K_r or the $m \times m$ grid as a minor.*

The notion of local treewidth was introduced by Eppstein [11] (see also [15]). The *local treewidth* of a graph G is

$$\mathbf{ltw}(G, r) = \max\{\mathbf{tw}(G[N_G^r[v]]): v \in V(G)\}.$$

For a function $f: N \rightarrow N$ we define the minor closed class of graphs of bounded local treewidth

$$\mathcal{L}(f) = \{G: \forall H \preceq G \forall r \geq 0, \mathbf{ltw}(H, r) \leq f(r)\}.$$

Also we say that a minor closed class of graphs $\mathcal{C}$ has bounded local treewidth if $\mathcal{C} \subseteq \mathcal{L}(f)$ for some function f .

Well known examples of minor closed classes of graphs of bounded local treewidth are planar graphs, graphs of bounded genus and graphs of bounded treewidth.

Many difficult graph problems can be solved efficiently when the input is restricted to graphs of bounded treewidth (see e.g., Bodlaender's survey [5]). Eppstein [11] made a step forward by proving that some problems like subgraph isomorphism and induced subgraph isomorphism can be solved in linear time on minor closed graphs of bounded local treewidth. Also the classical Baker's technique [4] for obtaining approximation schemes on planar graphs for different NP hard problems can be generalized to minor closed families of bounded local treewidth. (See [15] for a generalization of these techniques.)

An *apex graph* is a graph G such that for some vertex v (the *apex*), $G - v$ is planar. The following result is due to Eppstein [11].

Theorem 2 ([11]). *Let $\mathcal{F}$ be a minor-closed family of graphs. Then $\mathcal{F}$ is of bounded local treewidth if and only if $\mathcal{F}$ does not contain all apex graphs.*

3 Technical Lemma

In this section we prove the main technical lemma.

Lemma 1. *Let $G \in \mathcal{L}(f)$ be a graph containing the $m \times m$ grid H as a subgraph, $m > 2k^3$, where $k = 2f(2) + 2$. Then H contains the $(m/k^2 - 2k) \times (m - 2k)$ grid F as a subgraph such that for every vertex $v \in V(G)$, $|N_G[v] \cap V(F)| < k^2$, i.e. no vertex of G has $\geq k^2$ neighbors in F .*

Proof. We partition the grid H into k^2 subgraphs $H_1, H_2, \dots, H_{k^2}$. Each subgraph H_i is the $m/k^2 \times m$ grid induced by columns $1 + (i - 1)m/k^2, 2 + (i - 1)m/k^2, \dots, im/k^2$, $i \in \{1, 2, \dots, k^2\}$. Every grid H_i contains *inner* and *outer* parts. Inner part $\text{Inn}(H_i)$ is the $(m/k^2 - 2k) \times (m - 2k)$ grid obtained from H_i by removing k outer rows and columns. (See Fig. 1.)

For the sake of contradiction, suppose that every grid $\text{Inn}(H_i)$ contains a set of vertices S_i of cardinality $\geq k^2$ dominated by some vertex of G . We claim that

$$\begin{aligned} &H \text{ contains as a contraction the } k \times k^2 \text{ grid } T \text{ such that in a graph} \\ &G_T \text{ obtained from } G \text{ by contracting } H \text{ to } T \text{ for every column } C \text{ of } T \\ &\text{there is a vertex } v \in V(G_T) \text{ such that } N_{G_T}[v] \supseteq C. \end{aligned} \quad (1)$$

Before proving (1) let us explain why this claim brings us to a contradiction. Let T be a grid satisfying (1). Suppose first that there is a vertex v of G_T that dominates (in G_T) all vertices of at least k columns of T . Then these columns are the columns of a $k \times k$ grid which is a contraction of T . Thus G_T can be contracted to a graph of diameter 2 containing the $k \times k$ grid as a subgraph. This contraction has treewidth $\geq k$.

If there is no such vertex v , then there is a set D of k vertices $v_1, v_2, \dots, v_k$ of G_T such that every vertex $v_i \in D$ dominates all vertices of some column of T .

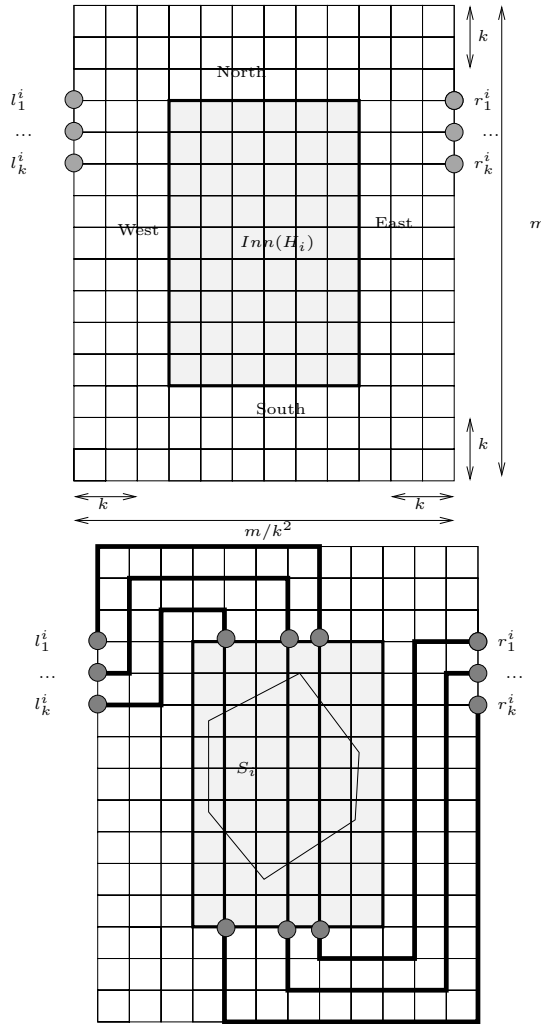


Fig. 1. Grid H_i and vertex disjoint paths connecting vertices $l_1^i, l_2^i, \dots, l_k^i$ with $r_1^i, r_2^i, \dots, r_k^i$.

Let $v_1, v_2, \dots, v_l$, $l \leq k$, be the vertices of D that are in T . Then T contains as a subgraph the $k/2 \times k/2$ grid P such that at least $k - l/2 \geq k/2$ vertices of D are outside P . Let us call these vertices D' . Every vertex of D' is outside P and dominates some column of P . By contracting all columns of P into one column we obtain $k/2$ vertices and each of these $k/2$ vertices is adjacent to all vertices of D' . Thus G contains the complete bipartite graph $K_{k/2, k/2}$ as a minor. $K_{k/2, k/2}$ has diameter 2 and treewidth $k/2$. In both cases we have that G contains a minor

of diameter ≤ 2 and of treewidth $\geq k/2 > f(2)$. Therefore $G \notin \mathcal{L}(f)$ which is a contradiction.

The remaining proof of the technical lemma is devoted to the proof of (1).

For every $i \in \{1, 2, \dots, k^2\}$, in the outer part of H_i we distinguish k vertices $l_1^i, l_2^i, \dots, l_k^i$ with coordinates $(k+1, 1), (k+2, 1), \dots, (2k, 1)$ and k vertices $r_1^i, r_2^i, \dots, r_k^i$ with coordinates $(k+1, m/k^2), (k+2, m/k^2), \dots, (2k, m/k^2)$. (See Fig. 1.)

We define *west (east) border* of $\text{Inn}(H_i)$ as the column of $\text{Inn}(H_i)$ which is the subcolumn of the $(k+1)$ st $((m/k^2 - k)$ th) column of H_i . *North (south) border* of $\text{Inn}(H_i)$ is the row of $\text{Inn}(H_i)$ that is subrow of the $(k+1)$ st $((m - k)$ th) row in H_i .

By assumption, every set S_i contains at least k^2 vertices in $\text{Inn}(H_i)$. Thus there are either k columns, or k rows of $\text{Inn}(H_i)$ such that each of these columns or rows has at least one vertex from S_i . This yields that there are k vertex disjoint paths either connecting north with south borders, or east with west borders and such that every path contains at least one vertex of S_i .

The subgraph of H_i induced by the first k columns and the first k rows is k -connected and by Menger's Theorem, for any k vertices of the west border of $\text{Inn}(H_i)$ (for any k vertices of the north border) there are k vertex disjoint paths connecting these vertices to the vertices $l_1^i, l_2^i, \dots, l_k^i$. By similar arguments any k vertices of the south border (east border) can be connected by k vertex disjoint paths with vertices $r_1^i, r_2^i, \dots, r_k^i$. (See Fig. 1.)

We conclude that for every $i \in \{1, 2, \dots, k^2\}$ there are k vertex disjoint paths in H_i with endpoints in $l_1^i, l_2^i, \dots, l_k^i$ and $r_1^i, r_2^i, \dots, r_k^i$ such that each path contains at least one vertex of S_i . Gluing these paths by adding edges (r_j^i, l_j^{i+1}) , $i \in \{1, 2, \dots, k^2 - 1\}$, $j \in \{1, 2, \dots, k\}$, we construct k vertex disjoint paths $P_1, P_2, \dots, P_k$ in H such that for every $j \in \{1, 2, \dots, k\}$

- P_j contains vertices $l_j^1, r_j^1, l_j^2, r_j^2, \dots, l_j^{k^2}, r_j^{k^2}$,
- For every $i \in \{1, 2, \dots, k^2\}$ P_j contains a vertex from S_i .

The subgraph of G induced by the paths $P_1, P_2, \dots, P_k$ contains as a contraction a grid T satisfying (1). This grid can be obtained by contracting edges of P_j , $j \in \{1, 2, \dots, k\}$ in such way, that at least one vertex of S_i of the subpath of P_j between vertices l_j^i and r_j^i is mapped to l_j^i . This grid has k^2 columns and each of the k^2 columns of T is dominated by some vertex of G_T . This concludes the proof of (1) and the lemma follows.

Corollary 1. *Let $G \in \mathcal{L}(f)$ be a graph containing the $m \times m$, $m > 2k^3$, where $k = 2f(2) + 2$, grid H as a minor. Then every dominating set of G is of size $> \frac{m^2}{k^4}$.*

Proof. Assume that G has a dominating set of size d . G contains as a contraction a graph G' such that G' contains H as a subgraph. Notice that G' also has a

dominating set of size d . By Lemma 1, H contains the $(m/k^2 - 2k) \times (m - 2k)$ grid F as a subgraph such that no vertex of G' has $\geq k^2$ neighbors in F . Thus

$$d \geq \frac{(m/k^2 - 2k) \times (m - 2k)}{k^2 + 1} > \frac{m^2}{k^4}.$$

4 Main Theorem

Theorem 3. *Let $\mathcal{F}$ be a minor-closed family of graphs. Then $\mathcal{F}$ has the domination-treewidth property if and only if $\mathcal{F}$ is of bounded local treewidth.*

Proof. In one direction the proof follows from Theorem 2. The apex graphs A_i , $i = 1, 2, 3, \dots$ obtained from the $i \times i$ grid by adding a vertex v adjacent to all vertices of the grid have a dominating set of size 1, diameter ≤ 2 and treewidth $\geq i$. So a minor closed family of graphs with domination-treewidth property cannot contain all apex graphs and hence it is of bounded local treewidth.

In the opposite direction the proof follows from the following claim

Claim. For any function $f: N \rightarrow N$ and any graph $G \in \mathcal{L}(f)$ with dominating set of size d , we have that $\text{tw}(G) = 2^{O(\sqrt{d} \log d)}$.

Let $G \in \mathcal{L}(f)$ be a graph of treewidth $m^{4r^2(m+2)}$ and with dominating set of size d . Let $r = f(1) + 2$ and $k = 2f(2) + 2$. Then G has no complete graph K_r as a minor. By Theorem 1, G contains the $m \times m$ grid H as a minor and by Corollary 1 $d \geq \frac{m^2}{k^4}$. Since k and r are constants depending only on f , we conclude that $m = O(\sqrt{d})$ and the claim and thus the theorem follows.

5 Algorithmic Consequences and Concluding Remarks

By general results of Frick & Grohe [14] the dominating set problem is fixed parameter tractable on minor-closed graph families of bounded local treewidth. However Frick & Grohe's proof is not constructive. It uses a transformation of first-order logic formulas into a 'local formula' according to Gaifman's theorem and even the complexity of this transformation is unknown.

Theorem 3 yields a constructive proof of the fact that the dominating set problem is fixed parameter tractable on minor-closed graph families of bounded local treewidth. It implies a fixed parameter algorithm that can be constructed as follows.

Let G be a graph from $\mathcal{L}(f)$. We want to check if G has a dominating set of size d . We put $r = f(1) + 2$ and $k = 2f(2) + 2$. First we check if the treewidth of G is at most $(\sqrt{d}k^2)^{4r^2(\sqrt{d}k^2+2)}$. This step can be performed by Amir's algorithm [3], which for a given graph G and integer ω , either reports that the treewidth of G is at least ω , or produces a tree decomposition of width at most $3\frac{2}{3}\omega$ in time $O(2^{3.698\omega} n^3 \omega^3 \log^4 n)$. Thus by using Amir's algorithm we can either compute a tree decomposition of G of size $2^{O(\sqrt{d} \log d)}$ in time $2^{2^{O(\sqrt{d} \log d)}} n^{3+\epsilon}$, or conclude that the treewidth of G is more than $(\sqrt{d}k^2)^{4r^2(\sqrt{d}k^2+2)}$.

- If the algorithm reports that $\text{tw}(G) > (\sqrt{d}k^2)^{4r^2(\sqrt{d}k^2+2)}$ then by Theorem 1 (G contains no K_r), G contains the $\sqrt{d}k^2 \times \sqrt{d}k^2$ grid as a minor. Then Corollary 1 implies that G has no dominating set of size d .
- Otherwise we perform a standard dynamic programming to compute dominating set. It is well known that the dominating set of a graph with a given tree decomposition of width at most ω can be computed in time $O(2^{2\omega}n)$ [1]. Thus this step can be implemented in time $2^{2^{O(\sqrt{d} \log d)}}n$.

We conclude with the following theorem.

Theorem 4. *There is an algorithm such that, for every minor-closed family $\mathcal{F}$ of bounded local treewidth and a graph $G \in \mathcal{F}$ on n vertices and an integer d , either computes a dominating set of size $\leq d$, or concludes that there is no such a dominating set. The running time of the algorithm is $2^{2^{O(\sqrt{d} \log d)}}n^{O(1)}$.*

Finally, some questions. For planar graphs and for some extensions it is known that for any graph G from this class with dominating set of size $\leq d$, we have $\text{tw}(G) = O(\sqrt{d})$. It is tempting to ask if the same holds for all minor-closed families of bounded local treewidth. This will provide subexponential fixed parameter algorithms on graphs of bounded local treewidth for the dominating set problem.

Another interesting and prominent graph class is the class of graphs containing no minor isomorphic to some fixed graph H . Recently Flum & Grohe [12] showed that parameterized versions of the dominating set problem is fixed-parameter tractable when restricted to graph classes with an excluded minor. Our result shows that the technique based on the dominating-treewidth property can not be used for obtaining constructive algorithms for the dominating set problem on excluded minor graph families. So constructing fast fixed parameter algorithms for these graph classes requires fresh ideas and is an interesting challenge.

Addendum

Recently we were informed (personal communication) that a result similar to the one of this paper was also derived independently (with a different proof) by Erik Demaine and MohammadTaghi Hajiaghayi.

Acknowledgement. The last author is grateful to Maria Satratzemi for technically supporting his research at the Department of Applied Informatics of the University of Macedonia, Thessaloniki, Greece.

References

1. J. ALBER, H. L. BODLAENDER, H. FERNAU, T. KLOKS, AND R. NIEDERMEIER, *Fixed parameter algorithms for dominating set and related problems on planar graphs*, Algorithmica, 33 (2002), pp. 461–493.

2. J. ALBER, H. FAN, M. FELLOWS, AND R. H. FERNAU NIEDERMEIER, *Refined search tree technique for dominating set on planar graphs*, in Mathematical Foundations of Computer Science—MFCS 2001, Springer, vol. 2136, Berlin, 2000, pp. 111–122.
3. E. AMIR, *Efficient approximation for triangulation of minimum treewidth*, in Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001), San Francisco, CA, 2001, Morgan Kaufmann Publishers, pp. 7–15.
4. B. S. BAKER, *Approximation algorithms for NP-complete problems on planar graphs*, J. Assoc. Comput. Mach., 41 (1994), pp. 153–180.
5. H. L. BODLAENDER, *A tourist guide through treewidth*, Acta Cybernetica, 11 (1993), pp. 1–23.
6. E. D. DEMAINE, M. HAJIAGHAYI, AND D. M. THILIKOS, *Exponential speedup of fixed parameter algorithms on $K_{3,3}$ -minor-free or K_5 -minor-free graphs*, in The 13th Annual International Symposium on Algorithms and Computation—ISAAC 2002 (Vancouver, Canada), Springer, Lecture Notes in Computer Science, Berlin, vol. 2518, 2002, pp. 262–273.
7. R. DIESTEL, *Graph theory*, vol. 173 of Graduate Texts in Mathematics, Springer-Verlag, New York, second ed., 2000.
8. R. DIESTEL, T. R. JENSEN, K. Y. GORBUNOV, AND C. THOMASSEN, *Highly connected sets and the excluded grid theorem*, J. Combin. Theory Ser. B, 75 (1999), pp. 61–73.
9. R. G. DOWNEY AND M. R. FELLOWS, *Parameterized complexity*, Springer-Verlag, New York, 1999.
10. J. ELLIS, H. FAN, AND M. FELLOWS, *The dominating set problem is fixed parameter tractable for graphs of bounded genus*, in The 8th Scandinavian Workshop on Algorithm Theory—SWAT 2002 (Turku, Finland), Springer, Lecture Notes in Computer Science, Berlin, vol. 2368, 2002, pp. 180–189.
11. D. EPPSTEIN, *Diameter and treewidth in minor-closed graph families*, Algorithmica, 27 (2000), pp. 275–291.
12. J. FLUM AND M. GROHE, *Fixed-parameter tractability, definability, and model-checking*, SIAM J. Comput.
13. F. V FOMIN AND D. M. THILIKOS, *Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-up*, Proceedings of the Fourteenth ACM-SIAM Symposium on Discrete Algorithms (SODA 2003), pp. 168–177.
14. M. FRICK AND M. GROHE, *Deciding first-order properties of locally tree-decomposable graphs*, J. ACM, 48 (2001), pp. 1184 – 1206.
15. M. GROHE, *Local tree-width, excluded minors, and approximation algorithms*. To appear in Combinatorica.
16. I. KANJ AND L. PERKOVIĆ, *Improved parameterized algorithms for planar dominating set*, in Mathematical Foundations of Computer Science—MFCS 2002, Springer, Lecture Notes in Computer Science, Berlin, vol. 2420, 2002, pp. 399–410.
17. N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. II. Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.
18. N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. V. Excluding a planar graph*, J. Comb. Theory Series B, 41 (1986), pp. 92–114.

Approximating Energy Efficient Paths in Wireless Multi-hop Networks^{*}

Stefan Funke, Domagoj Matijevic, and Peter Sanders

Max-Planck-Institut f. Informatik, 66123 Saarbrücken, Germany
`{funke,dmatijev,sanders}@mpi-sb.mpg.de`

Abstract. Given the positions of n sites in a radio network we consider the problem of finding routes between any pair of sites that minimize energy consumption and do not use more than some constant number k of hops. Known exact algorithms for this problem required $\Omega(n \log n)$ per query pair (p, q) . In this paper we relax the exactness requirement and only compute approximate $(1 + \epsilon)$ solutions which allows us to guarantee constant query time using linear space and $O(n \log n)$ preprocessing time. The dependence on ϵ is polynomial in $1/\epsilon$.

One tool we employ might be of independent interest: For any pair of points $(p, q) \in P \subseteq \mathbb{Z}^2$ we can report in constant time the cluster pair (A, B) representing (p, q) in a well-separated pair decomposition of P .

1 Introduction

Radio networks connecting a number of stations without additional infrastructure have recently gained considerable interest. Since the sites often have limited power supply, the energy consumption of communication is an important optimization criterion. We study this problem using the following simple geometric graph model: Given a set P of n points in $\mathbb{Z}^2$, we consider

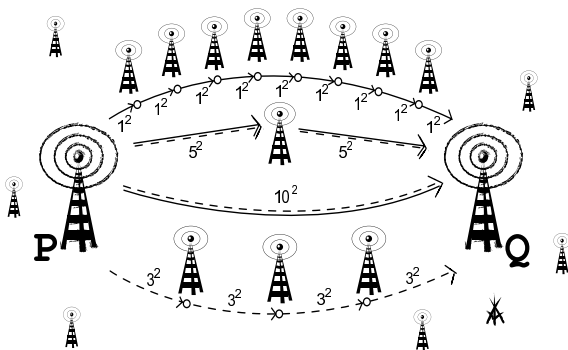


Fig. 1. A Radio Network and 9, 4, 2, 1-hop paths from P to Q with costs 9, 36, 50, 100

the complete graph $(P, P \times P)$ with edge weight $\omega(p, q) = |pq|^\delta$ for some constant $\delta > 1$ where $|pq|$ denotes the Euclidean distance between p and q . The objective is to find an approximate shortest path between two query points subject to the

^{*} This work was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

constraint that at most k edges of the graph are used in the path. For $\delta = 2$ the edge weights reflect the exact energy requirement for free space communication. For larger values of δ (typically between 2 and 4), we get a popular heuristic model for absorption effects [Rap96,Pat00]. Limiting the number of ‘hops’ to k can account for the distance independent overhead for using intermediate nodes. For a model with node dependent overheads refer to Section 4.

Our main result is a data structure that uses linear space and can be built in time $O(n \log n)$ for any constants k , $\delta > 1$, and $\epsilon > 0$. In constant time it allows to compute k -hop paths between arbitrary query points that are within a factor $(1 + \epsilon)$ from optimal. When k , δ , and ϵ are considered variables, the query time remains constant and the preprocessing time is bounded by a polynomial in k , δ , and $1/\epsilon$.

The algorithm has two main ingredients that are of independent interest. The first part, discussed in Section 2, is based on the observation that for approximately optimal paths it suffices to compute a shortest path for a constant size subset of the points — one point for each square cell in some grid that depends on the query points. This subset can be computed in time $O(\log n)$ using well known data structures supporting (approximate) quadratic range queries [BKOS,AM00]. These data structures and in particular their space requirement are independent of k , δ , and ϵ . Some variants even allow insertion and deletion of points in $O(\log n)$ time. Section 3 discusses the second ingredient. Well separated pair decompositions [CK92] allow us to answer arbitrary approximate path queries by precomputing a linear number of queries. We develop a way to access these precomputed paths in constant time using hashing. This technique is independent of path queries and can be used for retrieving any kind of information stored in well separated pair decompositions. Section 4 discusses further generalizations and open problems.

This extended abstract omits most proofs which can be found in the long version of the paper.

Related Work. Chan, Efrat, and Har-Peled [EH98,CE01] observe that for $\omega(p, q) = f(|pq|^\delta)$ and $\delta \geq 2$, exact geometric shortest paths are equivalent to shortest paths in the Delaunay triangulation of P , i.e., optimal paths can be computed in time $O(n \log n)$. Note that this approach completely collapses for k hop paths because most Delaunay edges are very short. The main contribution of Chan et al. is a sophisticated $O(n^{4/3+\gamma})$ time algorithm for computing exact geometric shortest paths for monotone cost functions $\omega(p, q) = f(|pq|)$ where γ is any positive constant. For quadratic cost functions with offsets $\omega(p, q) = |pq|^2 + C$, Beier, Sanders, and Sivasadan reduce that to $O(n^{1+\gamma})$, to $O(kn \log n)$ for k -hop paths, and to $O(\log n)$ time queries for *two hop* paths using linear space and $O(n \log n)$ time preprocessing. The latter result is very simple, it uses Voronoi diagrams and an associated point location data structure.

Thorup and Zwick [ThoZwi01] show that for general graphs and unrestricted k , it is impossible to construct a distance oracle which answers queries $2a - 1$ approximatively using space $o(nn^{1/a})$.

2 Fast Approximate k -Hop Path Queries

We consider the following problem: Given a set P of n points in $\mathbb{Z}^2$ and some constant k , report for a given query pair of points $p, q \in P$, a polygonal path $\pi = \pi(p, q) = v_0 v_1 v_2 \dots v_l$, with vertices $v_i \in P$ and $v_0 = p, v_l = q$ which consists of at most k segments, i.e. $l \leq k$, such that its weight $\omega(\pi) = \sum_{0 \leq i < l} \omega(v_i, v_{i+1})$ is minimized. By $\pi_{opt} = \pi_{opt}(p, q)$ we denote an optimal path from p to q under this criterion.

In the following we assume that the weight function ω is of the form $\omega(a, b) = |ab|^\delta$ with $\delta > 1$ (the case $\delta \leq 1$ is trivial as we just need to connect p and q directly by one hop).

2.1 Preliminaries

Before we introduce our procedure for reporting approximate k -hop paths, we need to refer to some standard data structures from Computational Geometry which will be used in our algorithm.

Theorem 1 (Exact Range Query). *Given a set P of n points in $\mathbb{Z}^2$ one can build a data structure of size $O(n \log n)$ in time $O(n \log n)$ which for a given axis aligned query rectangle $R = [x_l, x_u] \times [y_l, y_u]$ reports in $O(\log n)$ time either that R contains no point or outputs a point $p \in P \cap R$.*

The data structure can be maintained dynamically such that points can be inserted and deleted in $O(\log n \log \log n)$ amortized time. The preprocessing time then increases to $O(n \log n \log \log n)$ and the query time to $O(\log n \log \log n)$. All the $\log \log n$ factors can be removed if only either insertions or deletions are allowed.

In fact, the algorithm we will present will also work with an approximate range reporting data structure such as the one presented in [AM00, AM98]. The part of their result relevant for us can be stated in the following theorem:

Theorem 2 (Approximate Range Query). *Given a set P of n points in $\mathbb{Z}^2$ one can build a data structure of size $O(n)$ in time $O(n \log n)$ which for a given axis aligned query rectangle $R = [x_l, x_u] \times [y_l, y_u]$ with diameter ω reports in $O(\log n + \frac{1}{\alpha})$ time either that the rectangle $R' = [x_l + \alpha\omega, x_u + \alpha\omega] \times [y_l + \alpha\omega, y_u + \alpha\omega]$ contains no point or outputs a point $p \in P \cap R'$.*

The data structure can be maintained dynamically such that points can be inserted and deleted in $O(\log n)$ time.

Basically this approximate range searching data structure works well if the query rectangle is fat; and since our algorithm we present in the next section will only query square rectangular regions, all the results in [AM00] and [AM98] apply. In fact we do not even need α to be very small, $\alpha = 1$ turns out to be OK. So the use of an approximate range searching data structure helps us to get rid of the $\log n$ factor in space and some $\log \log n$ factors for the dynamic version. But to keep presentation simple we will assume for the rest of this paper that we have an exact range searching data structure at hand.

2.2 Computing Approximate k -Hop Paths for Many Points

We will now focus on how to process a k -hop path query for a pair of points p and q assuming that we have already constructed the data structure for orthogonal range queries (which can be done in time $O(n \log n)$).

Lemma 1. *For the optimal path π_{opt} connecting p and q we have $\frac{|pq|^\delta}{k^{\delta-1}} \leq |\pi_{\text{opt}}| \leq |pq|^\delta$.*

Definition 1. *We define the axis-aligned square of side-length l centered at the midpoint of a segment pq as the frame of p and q , $F(pq, l)$.*

Lemma 2. *The optimal path π_{opt} connecting p and q lies within the frame $F(pq, k^{(\delta-1)/\delta}|pq|)$ of p and q .*

We are now armed to state our algorithm to compute a k -hop path which is a $(1 + \epsilon)$ approximation to the optimal k -hop path from p to q .

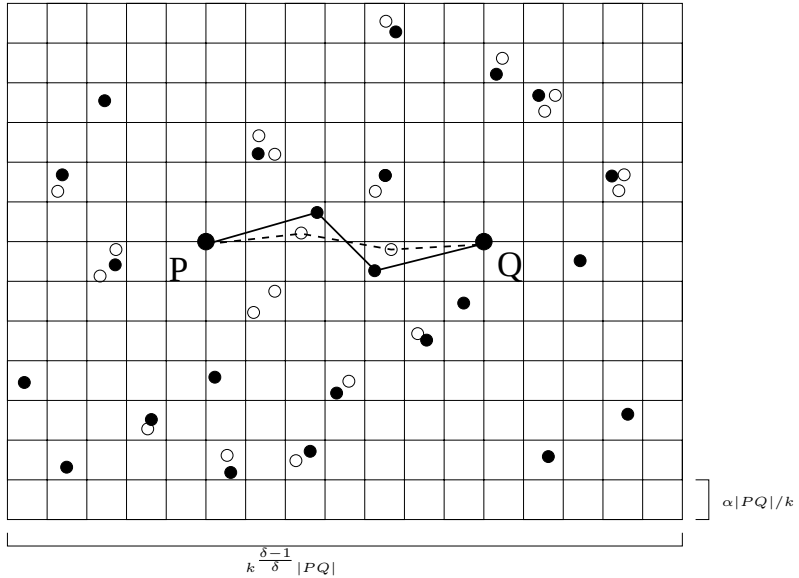


Fig. 2. 3-hop-query for P and Q : representatives for each cell are denoted as solid points, the optimal path is drawn dotted, the path computed by the algorithm solid

k -Hop-Query(p, q, ϵ)

1. Put a grid of cell-width $\alpha \cdot |pq|/k$ on the frame $F(pq, k^{(\delta-1)/\delta}|pq|)$ with $\alpha = \frac{1}{4\sqrt{2}} \cdot \frac{\epsilon}{\delta}$.

2. For each grid cell C perform an orthogonal range query to either certify that the cell is empty or report one point inside which will serve as a representative for C .
3. Compute the optimal k -hop path $\pi(p, q)$ with respect to all representatives and $\{p, q\}$.
4. Return $\pi(p, q)$

Please look at Figure 2.2 for a schematic drawing of how the algorithm computes the approximate k -hop path. It remains to argue about correctness and running time of our algorithm. Let us first consider its running time.

Lemma 3. *k -hop-Query(p, q, ϵ) can be implemented to return a result in time $O(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2} \cdot T_R(n) + T_{k,\delta}(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2}))$, where $T_R(n)$ denotes the time for one 2-dimensional range query on the original set of n points and $T_{k,\delta}(x)$ denotes the time for the exact computation of a minimal k -hop path for one pair amongst x points under the weight function $\omega(pq) = |pq|^\delta$.*

Let us now turn to the correctness of our algorithm, i.e. for any given ϵ , we want to show that our algorithm returns a k -hop path of weight at most $(1 + \epsilon)$ times the weight of the optimal path. We will show that only using the representatives of all the grid cells there exists a path of at most this weight. In the following we assume that the optimal path π_{opt} consists of a sequence of points $p_0 p_1 \dots p_j$, $j \leq k$ and $l_i = |p_{i-1} p_i|$.

Before we get to the actual proof of this claim, we need to state a small technical lemma.

Lemma 4. *For any $\delta > 1$ and $l_i, \xi > 0$ the following inequality holds*

$$\frac{\sum_{i=1}^k (l_i + \xi)^\delta}{\sum_{i=1}^k l_i^\delta} \leq \left(\frac{\sum_{i=1}^k (l_i + \xi)}{\sum_{i=1}^k l_i} \right)^\delta$$

Proof. Follows from Minkowski's and Hölder's inequalities.

Lemma 5. *k -hop-Query(p, q, ϵ) computes a k -hop path from p to q of weight at most $(1 + \epsilon)\omega(\pi_{\text{opt}}(p, q))$ for $0 < \epsilon \leq 1$.*

Proof. (Outline) Look at the 'detours' incurred by snapping the nodes of the optimal paths to the appropriate representative points and bound the overall 'detour' using Lemma 4.

2.3 Computing Optimal k -Hop Paths for Few Points

In our approximation algorithm we reduced the problem of computing an approximate k -hop path from p to q to one exact k -hop path computation of a small, i.e. constant number of points (only depending on k, δ and ϵ). Still, we have not provided a solution for this problem yet. In the following we will present first a generic algorithm which works for all possible δ and then quickly review the exact algorithm presented by [BSS02] which only works for the case $\delta = 2$, though.

Layered Graph Construction. We can consider almost the complete graph with all edge weights explicitly stored (except for too long edges, which cannot be part of the optimal solution) and then use the following construction:

Lemma 6. *Given a connected graph $G(V, E)$ with $|V| = n$, $|E| = m$ with weights on the edges and one distinguished node $s \in V$, one can compute for all $p \in V - \{s\}$ the path of minimum weight using at most k edges in time $O(km)$.*

Proof. We assume that the graph G has self-loop edges (v, v) with assigned weight 0. Construct $k + 1$ copies $V^{(0)}, V^{(1)}, \dots, V^{(k)}$ of the vertex set V and draw a directed edge $(v^{(i)}, w^{(i+1)})$ iff $(v, w) \in E$ with the same weight. Compute the distances from $s^{(0)}$ to all other nodes in this layered, acyclic graph. This takes time $O(km)$ as each edge is relaxed only once. $\square$

So in our subproblem we can use this algorithm and the property that each representative has $O(\frac{\delta^2 k^2}{\epsilon^2})$ adjacent edges (all other edges are too long to be useful) to obtain the following corollary:

Corollary 1. *The subroutine of our algorithm to solve the exact k -hop problem on $O(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2})$ points can be solved in time $O(\frac{\delta^4 \cdot k^{(7\delta-2)/\delta}}{\epsilon^4})$ for arbitrary δ, ϵ .*

Reduction to Nearest Neighbor. In [BSS02] the authors presented an algorithm which for the special case $\delta = 2$ computes the optimal k -hop path in time $O(kn \log n)$ by dynamic programming and an application of geometric nearest neighbor search structures to speed up the update of the dynamic programming table. Applied to our problem we get the following corollary:

Corollary 2. *The subroutine of our algorithm to solve the exact k -hop problem on $O(\frac{k^3}{\epsilon^2})$ points can be solved in time $O(\frac{k^5}{\epsilon^2} \cdot \log \frac{k}{\epsilon})$ if $\delta = 2$.*

2.4 Summary

Let us summarize our general result in the following Theorem (we give the bound for the case where an approximate nearest neighbor query data structure as mentioned in Theorem 2 is used).

Theorem 3. *We can construct a dynamic data structure allowing insertions and deletions with $O(n)$ space and $O(n \log n)$ preprocessing time such that $(1 + \epsilon)$ approximate minimum k -hop path queries under the metric $\omega(p, q) = |pq|^\delta$ can be answered in time $O(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2} \cdot \log n + \frac{\delta^4 \cdot k^{(7\delta-2)/\delta}}{\epsilon^4})$.*

The query time does not change when using exact range query data structures, only space, preprocessing and update times get slightly worse (see Theorem 1). For the special case of $\delta = 2$ we obtain a slightly improved query time of $O(\frac{\delta^2 \cdot k^{(4\delta-2)/\delta}}{\epsilon^2} \cdot \log n + \frac{\delta^2 \cdot k^{(5\delta-2)/\delta}}{\epsilon^2} \cdot \log \frac{\delta k}{\epsilon})$.

3 Precomputing Approximate k -Hop Paths for Constant Query Time

In the previous section we have seen how to answer a (p, q) query in $O(\log n)$ time (considering k, δ, ϵ as constants). Standard range query data structures were the only precomputed data structures used. Now we explain how additional precomputation can further reduce the query time. We show how to precompute a linear number of k -hop paths, such that for every (p, q) , a slight modification of one of these precomputed paths is a $(1 + \epsilon)(1 + 2\psi)^2$ approximate k -hop path and such a path can be accessed in constant time. Here $\psi > 0$ is the error incurred by the use of these precomputed paths and can be chosen arbitrarily small (the size of the well-separated pair decomposition then grows, though).

3.1 The Well-Separated Pair Decomposition

We will first briefly introduce the so-called *well-separated pair decomposition* due to Callahan and Kosaraju ([CK92]).

The *split-tree* of a set P of points in $\mathbb{R}^2$ is the tree constructed by the following recursive algorithm:

SplitTree(P).

1. if $\text{size}(P)=1$ then return $\text{leaf}(P)$
2. partition P into sets P_1 and P_2 by halving its minimum enclosing box $R(P)$ along its longest dimension
3. return a node with children ($\text{SplitTree}(P_1)$, $\text{SplitTree}(P_2)$)

Although such a tree might have linear depth and therefore a naive construction as above takes quadratic time, Callahan and Kosaraju in [CK92] have shown how to construct such a binary tree in $O(n \log n)$ time. With every node of that tree we can conceptually associate the set A of all points contained in its subtree as well as their minimum enclosing box $R(A)$. By $r(A)$ we denote the radius of the minimum enclosing disk of $R(A)$. We will also use A to denote the node associated with the set A if we know that such a node exists.

For two sets A and B associated with two nodes of a split tree, $d(A, B)$ denotes the distance between the centers of $R(A)$ and $R(B)$ respectively. A and B are said to be *well-separated* if $d(A, B) > sr$, where r denotes the radius of

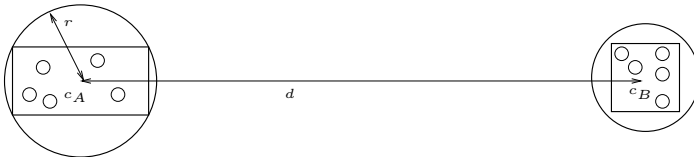


Fig. 3. Clusters A and B are 'well-separated' if $d > s \cdot r$

the larger of the two minimum enclosing balls of $R(A)$ and $R(B)$ respectively. s is called the *separation constant*.

In [CK92], Callahan and Kosaraju present an algorithm which, given a split tree of a point set P with $|P| = n$ and a separation constant s , computes in time $O(n(s^2 + \log n))$ a set of $O(n \cdot s^2)$ additional *blue* edges for the split tree, such that

- the point sets associated with the endpoints of a blue edge are well-separated with separation constant s .
- for any pair of leaves (a, b) , there exists exactly one blue edge that connects two nodes on the paths from a and b to their lowest common ancestor $lca(a, b)$ in the split tree

The split tree together with its additional blue edges is called the *well-separated pair decomposition (WSPD)*.

3.2 Using the WSPD for Precomputing Path Templates

In fact the WSPD is exactly what we need to efficiently precompute k -hop paths for all possible $\Theta(n^2)$ path queries. So we will use the following preprocessing algorithm:

1. compute a well-separated pair decomposition of the point set with $s = k^{(\delta-1)/\delta} \cdot 8\delta \cdot \frac{1}{\psi}$
2. for each *blue* edge compute a $(1 + \epsilon)$ -approximation to the lightest k -hop path between the centers of the associated bounding boxes

At query time, for a given query pair (p, q) , it remains to find the unique blue edge (A, B) which links a node of the path from p to $lca(p, q)$ to a node of the path from q to $lca(p, q)$. We take the precomputed k -hop path associated with this blue edge, replace its first and last node by s and t respectively and return this modified path.

In the following we will show that the returned path is indeed a $(1 + \epsilon)(1 + 2\psi)^2$ approximation of the lightest k -hop path from p to q . Later we will also show that this path can be found in constant time. For the remainder of this section let $\pi_{\text{opt}}^P(x, y)$ denote the optimal k -hop path between two points x, y not necessarily in P such that all hops have starting and end point in P (except for the first and last hop). We first start with a lemma which formalizes the intuition that the length of an optimal k -hop path does not change much when perturbing the query points slightly.

Lemma 7. *Given a set of points P and two pairs of points (a, b) and (a', b') with $d(a, b) = d$ and $d(a, a') \leq c$, $d(b, b') \leq c$ with $c \leq \frac{\psi d}{k^{(\delta-1)/\delta} \cdot 4\delta}$, then we have $\omega(\pi_{\text{opt}}^P(a', b')) \leq (1 + 2\psi)\omega(\pi_{\text{opt}}^P(a, b))$.*

The following corollary of the above Lemma will be used later in the proof:

Corollary 3. *Given a set of points P and two pairs of points (a, b) and (a', b') with $d(a, b) = d$ and $d(a, a') \leq c$, $d(b, b') \leq c$ with $c \leq \frac{\psi d}{k^{(\delta-1)/\delta} \cdot 8\delta}$, then we have $\omega(\pi_{opt}^P(a', b')) \leq (1 + 2\psi)\omega(\pi_{opt}^P(a, b))$ as well as $\omega(\pi_{opt}^P(a, b)) \leq (1 + 2\psi)\omega(\pi_{opt}^P(a', b'))$.*

Proof. Clearly the first claim holds according to Lemma 7. For the second one observe that $d' = |a'b'| \geq d - 2 \cdot c$ and then apply the Lemma again. $\square$

Applying this Corollary, it is now straightforward to see that the approximation ratio of the modified template path is $(1 + 2\psi)^2(1 + \epsilon)$.

Lemma 8. *Given a well separated pair decomposition of a point set $P \subset \mathbb{Z}^2$ with separation constant $s = \frac{k^{(\delta-1)/\delta} \cdot 8\delta}{\psi}$, the path $\pi(p, q)$ returned for a query pair (p, q) is a $(1 + 2\psi)^2(1 + \epsilon)$ approximate k -hop path from p to q .*

We leave it to the reader to figure out the right choice for ψ and ϵ to obtain an arbitrary approximation quality of $(1 + \phi)$, but clearly $\psi, \epsilon \in \Omega(\phi)$.

3.3 Retrieving Cluster Pairs for Query Points in $O(1)$ Time

In the previous paragraphs we have shown that using properties of the well-separated pair decomposition, it is possible to compute $O(n)$ 'template paths' such that for any query pair (s, t) out of the $\Omega(n^2)$ possible query pairs, there exists a good template path which we can modify to obtain a good approximation to the lightest k -hop path from s to t . Still, we have not shown yet how to determine this good template path for a given query pair (s, t) in constant time. We note that the following description does not use any special property of our original problem setting, so it may apply to other problems, where the well-separated pair decomposition can be used to encode in $O(n)$ space sufficient information to cover a query space of $\Omega(n^2)$ size.

Gridding the Cluster Pairs. The idea of our approach is to round the centers c_A, c_B of a cluster pair (A, B) which is part of the WSPD to canonical grid points $\widetilde{c}_A, \widetilde{c}_B$ such that for any query pair (s, t) we can determine $\widetilde{c}_A, \widetilde{c}_B$ in constant time. Furthermore we will show that there is only a constant number of cluster pairs (A', B') which have their cluster centers rounded to the same grid positions $\widetilde{c}_A, \widetilde{c}_B$, so well-known hashing techniques can be used to store and retrieve the respective blue edge and also some additional information $I_{(A, B)}$ (in our case: the precomputed k -hop path) associated with that edge.

In the following we assume that we have already constructed a WSPD of the point set P with a separation constant $s > 4$. For any point $p \in \mathbb{Z}^2$, let $\text{snap}(p, w)$ denote the closest grid-point of the grid with cell-width w originated at $(0, 0)$ and let $H : \mathbb{Z}^4 \rightarrow (I \times E)^*$ denote a hash table data structure which maps pairs of integer points in the plane to a list of pairs consisting of some information type and a (blue) edge in the WSPD. Using universal hashing [CW79] this data structure has constant expected access time.

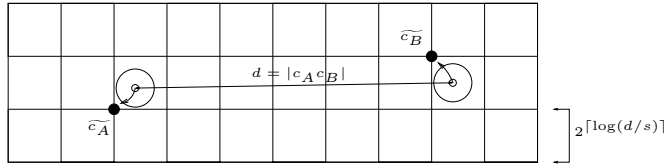


Fig. 4. Cluster centers c_A and c_B are snapped to closest grid points $\widetilde{c}_A$ and $\widetilde{c}_B$

Preprocessing.

- For every blue edge connecting clusters (A, B) in the split tree
 - $c_A \leftarrow \text{center}(R(A))$, $c_B \leftarrow \text{center}(R(B))$
 - $w \leftarrow |c_A c_B|/s$
 - $\widetilde{w} \leftarrow 2^{\lceil \log w \rceil}$
 - $\widetilde{c}_A \leftarrow \text{snap}(c_A, \widetilde{w})$
 - $\widetilde{c}_B \leftarrow \text{snap}(c_B, \widetilde{w})$
 - Append $((I_{(A,B)}, (A, B)))$ to $H[(\widetilde{c}_A, \widetilde{c}_B)]$

Look at Figure 4 for a sketch of the preprocessing routine for one cluster pair (A, B) . Clearly this preprocessing step takes linear time in the size of the WSPD. So given a query pair (s, t) , how to retrieve the information $I_{(A,B)}$ stored with the unique cluster pair (A, B) with $s \in A$ and $t \in B$?

Query (p, q) .

- $w' \leftarrow |pq|/s$
- $\widetilde{w}_1 \leftarrow 2^{\lceil \log w' \rceil - 1}$
- $\widetilde{w}_2 \leftarrow 2^{\lceil \log w' \rceil}$
- $\widetilde{w}_3 \leftarrow 2^{\lceil \log w' \rceil + 1}$
- for grid-widths w_i , $i = 1, 2, 3$ and adjacent grid-points $\widetilde{c}_p, \widetilde{c}_q$ of p and q respectively
 - Inspect all items $(I_{(A,B)}, (A, B))$ in $H[(\widetilde{c}_p, \widetilde{c}_q)]$
 - * if $p \in A$ and $q \in B$ return $I_{(A,B)}$

In this description we call a grid-point $\widetilde{g}$ *adjacent* to a point p if $|\widetilde{g}p|_x, |\widetilde{g}p|_y < \frac{3}{2}\widetilde{w}$, where $|\cdot|_{x/y}$ denotes the horizontal/vertical distance. Clearly there are at most 9 *adjacent* points for any point p in a grid of width $\widetilde{w}$. In the remainder of this section we will show that this query procedure outputs the correct result (the unique $I_{(A,B)}$ with $s \in A$ and $t \in B$ such that (A, B) is blue edge in the WSPD) and requires only constant time. In the following we stick to the notation that $\widetilde{w} = 2^{\lceil \log |c_A c_B|/s \rceil}$, where c_A, c_B are the cluster centers of the cluster pair (A, B) we are looking for.

Lemma 9. *For $s > 4$, we have $\widetilde{w}_i = \widetilde{w}$ for some $i \in \{1, 2, 3\}$.*

This Lemma says that at some point the query procedure uses the correct grid-width as determined by c_A and c_B . Furthermore for any given grid-width and a pair of query points p and q , there are at most $9 \cdot 9 = 81$ pairs of adjacent

grid points to inspect. We still need to argue that given the correct grid-width $\tilde{w}$, the correct pair of grid points $(\tilde{c}_A, \tilde{c}_B)$ is amongst these ≤ 81 possible pairs of grid points that are inspected.

Lemma 10. *For $\tilde{w}_i = \tilde{w}$, $\tilde{c}_A$ and $\tilde{c}_B$ are amongst the inspected grid-points.*

The last thing to show is that only a constant number of cluster pairs (A, B) can be rounded during the preprocessing phase to a specific pair of grid positions $(\tilde{g}_1, \tilde{g}_2)$ and therefore we only have to scan a list of constant size that is associated with $(\tilde{g}_1, \tilde{g}_2)$. Before we can prove this, we have to cite a Lemma from the original work of Callahan and Kosaraju on the WSPD [CK92].

Lemma 11 (CK92). *Let C be a d -cube and let $S = \{A_1, \dots, A_l\}$ be a set of nodes in the split tree such that $A_i \cap A_j = \emptyset$ and $l_{\max}(p(A_i)) \geq l(C)/c$ and $R(A_i)$ overlaps C for all i . Then we have $l \leq (3c + 2)^d$.*

Here $p(A)$ denotes the parent of a node A in the split tree, $l_{\max}(A)$ the longest side of the minimum enclosing box of $R(A)$.

Lemma 12. *Consider a WSPD of a point set P with separation constant $s > 4$, grid width $\tilde{w}$ and a pair of grid points $(\tilde{g}_1, \tilde{g}_2)$. The number of cluster pairs (A, B) such that c_A and c_B are rounded to $(\tilde{g}_1, \tilde{g}_2)$ is $O(1)$.*

Proof. Follow from the previous Lemma, see the full version for details.

Putting everything together we get the main theorem of this section:

Theorem 4. *Given a well-separated pair decomposition of a point set P with separation constant $s > 4$. Then we can construct a data structure in space $O(n \cdot s^2)$ and construction time $O(n \cdot s^2)$ such that for any pair of points (p, q) in P we can determine the unique pair of clusters (A, B) that is part the well-separated pair decomposition with $p \in A$, $q \in B$ in constant time.*

Together with the results of the previous Section we obtain the following main result of our paper:

Theorem 5. *Given a set of points $P \subset \mathbb{Z}^2$, a distance function $\omega : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{R}^+$ of the form $\omega(p, q) = |pq|^\delta$, where $\delta \geq 1$ and $k \geq 2$ are constants, we can construct a data structure of size $O(\frac{1}{\epsilon^2} \cdot n)$ in preprocessing time $O(\frac{1}{\epsilon^4} \cdot n \log n + \frac{1}{\epsilon^6} \cdot n)$ such that for any query (p, q) from P , a $(1 + \epsilon)$ -approximate lightest k -hop path from p to q can be obtained in constant $O(1)$ time which does not depend on δ, ϵ, k .*

We also remark that there are techniques to maintain the well-separated pair decomposition dynamically, and so our whole construction can be made dynamic as well (see [CK95]).

4 Discussion

We have developed a data structure for constant approximate shortest path queries in a simple model for geometric graphs. Although this model is motivated by communication in radio networks, it is sufficiently simple to be of independent theoretical interest and possibly for other applications. For example, Chan, Efrat, and Har-Peled [EH98,CE01] use similar concepts to model the fuel consumption of airplanes routed between a set P of airports.

We can also further refine the model. For example, the above flight application would require more general cost functions. Here is one such generalization: If the cost of edge (p, q) is $|pq|^\delta + C_p$ for a node dependent cost offset C_p , our result remains applicable under the assumption of some bound on the offset costs. In Lemma 5 we would choose the cell representative as the node with minimum offset in the cell (this can be easily incorporated into the standard geometric range query data structures). The offset could model distance independent energy consumption like signal processing costs or it could be used to steer away traffic from devices with low battery power.

References

- [AM00] S. Arya and D. M. Mount: *Approximate range searching*, Computational Geometry: Theory and Applications, (17), 135-152, 2000
- [AM98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, A. Wu: *An optimal algorithm for approximate nearest neighbor searching*, Journal of the ACM, 45(6):891-923, 1998
- [BSS02] R. Beier, P. Sanders, N. Sivadasan: *Energy Optimal Routing in Radio Networks Using Geometric Data Structures* Proc. of the 29th Int. Coll. on Automata, Languages, and Programming, 2002.
- [BKOS] M. de Berg, M. van Krefeld, M. Overmars, O. Schwarzkopf: *Computational Geometry: Algorithms and Applications*, Springer, 1997
- [CK92] P.B. Callahan, S.R. Kosaraju: *A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields*, Proc. 24th Ann. ACM Symp. on the Theory of Computation, 1992
- [CK95] P.B. Callahan, S.R. Kosaraju: *Algorithms for Dynamic Closest Pair and n-Body Potential Fields*, Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithm, 1995
- [CW79] J.L. Carter and M.N. Wegman. Universal Classes of Hash Functions. *Journal of Computer and System Sciences*, 18(2):143-154, 1979
- [CE01] T. Chan and A. Efrat. Fly cheaply: On the minimum fuel consumption problem. *Journal of Algorithms*, 41(2):330-337, November 2001.
- [EH98] A. Efrat, S. Har-Peled: *Fly Cheaply: On the Minimum Fuel-Consumption Problem*, Proc. 14th ACM Symp. on Computational Geometry 1998.
- [MN90] K. Mehlhorn, S. Näher: *Dynamic Fractional Cascading*, Algorithmica (5), 1990, 215-241
- [Pat00] D. Patel. Energy in ad-hoc networking for the picoradio. Master's thesis, UC Berkeley, 2000.
- [Rap96] T. S. Rappaport. *Wireless Communication*. Prentice Hall, 1996.
- [ThoZwi01] M. Thorup and U. Zwick. Approximate Distance Oracles *Proc. of 33rd Symposium on the Theory of Computation 2001*.

Bandwidth Maximization in Multicasting

Naveen Garg¹, Rohit Khandekar^{1*}, Keshav Kunal¹, and
Vinayaka Pandit²

¹ Department of Computer Science & Engineering,
Indian Institute of Technology,
New Delhi 110016, India.

`{naveen, rohitk, keshav}@cse.iitd.ernet.in`

² IBM India Research Lab,
Block I, Indian Institute of Technology,
New Delhi 110016, India.
`pvinayak@in.ibm.com`

Abstract. We formulate bandwidth maximization problems in multicasting streaming data. Multicasting is used to stream data to many *terminals* simultaneously. The goal here is to maximize the bandwidth at which the data can be transmitted satisfying the capacity constraints on the links. A typical network consists of the *end-hosts* which are capable of duplicating data instantaneously, and the *routers* which can only forward the data. We show that if one insists that all the data to a terminal should travel along a single path, then it is NP-hard to approximate the maximum bandwidth to a factor better than 2. We also present a fast 2-approximation algorithm. If different parts of the data to a terminal can travel along different paths, the problem can be approximated to the same factor as the minimum Steiner tree problem on undirected graphs. We also prove that in case of a tree network, both versions of the bandwidth maximization problem can be solved optimally in polynomial time. Of independent interest is our result that the minimum Steiner tree problem on tree-metrics can be solved in polynomial time.

1 Introduction

Multicasting is a useful method of efficiently delivering the same data to multiple recipients in a network. The IP layer has long been considered the natural protocol layer to implement multicast. However concerns related to scalability, deployment etc. continue to dog it. In this context, researchers [5,7,8,2,3] have proposed an alternative architecture where multicast functionality is supported at the application layer by end-hosts. In the application layer multicast, data packets are replicated only at end-hosts and not at the routers in the network. Conceptually, end-hosts are viewed as forming an overlay network for supporting multicast.

One therefore needs to pick a suitable overlay network so as to minimize the performance penalty incurred due to duplicate data packets and long delays. The

* Partially supported by a fellowship from Infosys Technologies Ltd., Bangalore.

different application layer multicast schemes proposed either build a tree [3,8] or a mesh [2,5] as the overlay network. In this paper we will consider the setting where we are multicasting streaming data over a tree overlay network and the metric of interest to us is the throughput/bandwidth of the overlay network.

We model the network as an undirected graph on two kinds of nodes called *end-hosts* and *routers*. The end-hosts are capable of generating and replicating the traffic while the routers are used for forwarding the data. We assume that the terminals, the nodes that are required to receive the data, are end-hosts. The multicasting is done via a *multicast tree* in which the root first sends the streaming data to (one or more) end-hosts. These end-hosts, in turn, make multiple copies of the data instantaneously and forward it to other end-hosts. This continues till each terminal receives the data. Each link between a pair of nodes has a capacity which is the maximum bandwidth it can support in either directions put together. The objective is to maximize the bandwidth that can be routed to the terminals respecting the link capacities.

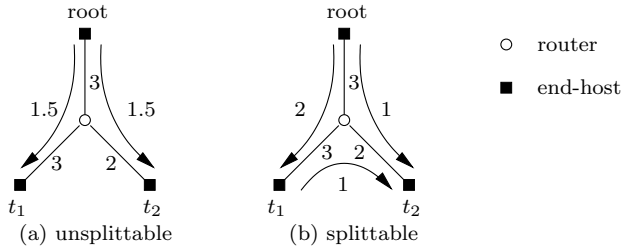


Fig. 1. The numbers on the links denote their capacities while the ones on the paths denote the bandwidth routed. (a) In unsplittable version, a maximum bandwidth of 1.5 is routed to t_1 and t_2 directly; (b) In splittable version, a bandwidth of 2 and 1 is routed to t_1 and t_2 resp. and t_1 forwards a bandwidth of 1 to t_2 .

We consider many variants of the bandwidth maximization problem. In the *unsplittable* bandwidth maximization problem (UBM), one desires that the data to any terminal be routed via a single path from the root. On the other hand, in the *splittable* bandwidth maximization problem (SBM), different parts of the data to a terminal can arrive along different paths from the root (see Figure 1). We prove that even if all end-hosts are terminals, it is NP-hard to approximate UBM to a factor less than 2. We reduce the NP-complete Hamiltonian cycle problem on 3-regular graphs [4] to it (Section 4.2). On the positive side, we present a very simple and efficient 2-approximation algorithm for UBM (Section 4.3). For the splittable case, we first observe that SBM is equivalent to finding a maximum fractional packing of *multicast trees* and prove that there is an α -approximation for SBM if and only if there is an α -approximation for the minimum Steiner tree problem on undirected graphs (Section 3). Our technique is similar to that of Jain et al. [6] who prove that there is an α -approximation for maximum fractional packing of Steiner trees if and only if there is an α -

approximation for the minimum Steiner tree problem on undirected graphs. Our results imply that if all the end-hosts are terminals, then SBM can be solved in polynomial time.

We consider yet another variant of the bandwidth maximization problem in which between any pair of end-hosts, the data can be routed only through a pre-specified set of paths. We can think of UBM and SBM problems with this restriction. We prove that SBM with this restriction can also be approximated to within the Steiner ratio, while if all end-hosts are terminals, then it can be solved in polynomial time.

If the network is a tree, UBM and SBM can be solved in polynomial time. First we reduce UBM to the problem of deciding whether a particular bandwidth can be routed to the terminals respecting the link capacities. This problem in turn is solved using dynamic programming to keep track of paths of the multicast tree that go in or out of various sub-trees. Overall, UBM can be solved in $O(n \log n)$ time on a tree on n nodes (Section 4.1). We present a polynomial time algorithm to solve the minimum Steiner tree problem on a tree-metric (Section 3.1). This implies that SBM can also be solved in polynomial time on trees. Since there is a unique simple path between any pair of end-hosts on the tree network, the pre-specified-path version is identical to the unrestricted version.

A summary of the results obtained in this paper follows.

		SBM		UBM	
		$T \subset H$	$T = H$	$T \subset H$	$T = H$
General Graph Networks	Arbitrary path	$\mathcal{S}$ -approx	Polytime	2-approx ($2 - \epsilon$)-hard	
	Prespecified paths	$\mathcal{S}$ -approx	Polytime	?	?
Tree Networks	Arbitrary path $\equiv$ Prespec. paths	Polytime		Polytime	

Here $\mathcal{S}$ denotes the factor to which the minimum Steiner tree problem can be approximated. The set of end-hosts is denoted by H , and the set of terminals by T . The question marks indicate the open problems.

2 Problem Formulation

In this section, we formulate the bandwidth maximization problem for multicasting streaming data. We are given an undirected graph $G = (H \cup R, E)$ in which the node-set is a disjoint union of the *end-hosts* H and the *routers* R . We call each element in E as a *link*. Each link $e \in E$ has *capacity* c_e which is the maximum bandwidth it can carry in both directions put together. We are also given a special node $r \in H$ called the *root*, and a set $T \subseteq H$ of *terminals*. Our goal is to multicast the same streaming data from r to all the terminals $t \in T$. The routers can forward data from an incoming link to an outgoing link; they cannot, however, make multiple copies of the data. The end-hosts can perform the function of routers as well as make duplicate copies of the data instantaneously.

Let $K_{|H|}$ be a complete graph on all end-hosts. Let Q be a tree in $K_{|H|}$ rooted at r , which spans T . Conceptually, the data is routed from the root to the terminals via the tree Q . An edge $(h, h') \in Q$ (where h is the parent of h') is realized in the actual network by a path $P_{hh'}$ from h to h' in G . Formally,

Definition 1. A multicast tree is a pair $M = (Q, P)$ where

- Q is a tree in $K_{|H|}$ rooted at r which spans $\{r\} \cup T$
- P is a mapping of each edge $(h, h') \in Q$, where h is a parent of h' , to a unique path $P_{h,h'} \subset G$ from h to h' .

For a link $e \in E$, let $p_{e,M} = |\{(h, h') \in Q | e \in P_{hh'}\}|$ denote the number of paths of M that go through $e \in E$. Observe that $p_{e,M} \leq |H| - 1$ because Q contains at most $|H| - 1$ edges. Clearly, if a bandwidth of λ is to be routed using M , the capacity constraint of each link $e \in E$ enforces that $\lambda \leq c_e/p_{e,M}$. We denote the maximum bandwidth which can be routed using M by $\lambda_M = \min_{e \in E} c_e/p_{e,M}$.

Unsplittable Bandwidth Maximization Problem (UBM)

- **Input:** $G = (H \cup R, E)$, $c : E \rightarrow \mathbb{R}^+$, $r \in H$ and $T \subseteq H$.
- **Output:** A multicast tree $M = (Q, P)$.
- **Goal:** To maximize the bandwidth λ_M .

Consider a single packet in the splittable bandwidth maximization problem. It reaches each terminal through a path from r . The union of the paths of a single packet to the terminals defines a multicast tree. In SBM, the bandwidth maximization can be thought of as packing as many multicast trees as possible without exceeding the capacity of any link. For each multicast tree M , we associate $\sigma_M \geq 0$ which denotes the bandwidth routed via M .

Splittable Bandwidth Maximization Problem (SBM)

- **Input:** $G = (H \cup R, E)$, $c : E \rightarrow \mathbb{R}^+$, $r \in H$ and $T \subseteq H$.
- **Output:** An assignment $\sigma_M \geq 0$ of the bandwidth to each multicast tree M such that for any link $e \in E$, we have the following capacity constraint satisfied: $\sum_M \sigma_M p_{e,M} \leq c_e$.
- **Goal:** To maximize the total bandwidth routed: $\sum_M \sigma_M$.

3 Splittable Bandwidth Maximization

The SBM problem on general graphs can be viewed naturally as a linear program. For each multicast tree M , we associate a real variable $\sigma_M \geq 0$ to indicate the bandwidth routed through M . This linear program can be viewed as a fractional packing of multicast trees. Since it has exponentially many variables, we also consider its dual.

Primal	Dual
$\begin{aligned} \max \quad & \sum_M \sigma_M \\ \text{s.t.} \quad & \sum_M p_{e,M} \sigma_M \leq c_e \quad \forall e \in E \\ & \sigma_M \geq 0 \quad \forall M \end{aligned}$	$\begin{aligned} \min \quad & \sum_{e \in E} c_e l_e \\ \text{s.t.} \quad & \sum_{e \in M} p_{e,M} l_e \geq 1 \quad \forall M \\ & l_e \geq 0 \quad \forall e \in E \end{aligned}$

The separation problem for the dual program is: Given a length assignment l on the links $e \in E$, determine if there exists a multicast tree M such that $\sum_{e \in M} p_{e,M} l_e < 1$. This can be done by computing the minimum length multicast tree $M^* = (Q^*, P^*)$ for the given length function, and comparing the length of M^* with 1. Observe that for any edge $(h, h') \in Q^*$, the length of the corresponding path $P_{h,h'}^*$ will be equal to the length of the shortest path (among the set of specified paths) between h and h' in G under length l . Thus to compute M^* , we consider the complete graph $K_{|H|}$ on the end-hosts and assign the edge (h, h') a length equal to the length of the shortest path (under lengths l) between h and h' in G . The tree Q^* is now the minimum Steiner tree spanning $\{r\} \cup T$.

Note that the metric on H is defined by shortest paths in graph G . We call such a metric, a G -metric. Formally, for a graph G and a subset H of nodes of G , a metric d on H is called a G -metric if there is a length function l on the links of G such that for any $h, h' \in H$, $d(h, h')$ equals the length of the shortest path between h and h' in G under the length function l . The following theorem holds.

Theorem 1. *Let $G = (H \cup R, E)$, $r \in H, T \subseteq H$ be an instance of SBM. There is a polynomial time α -approximation for SBM on this instance with any capacity function on E if and only if there is a polynomial time α -approximation for the minimum Steiner tree problem on H with a G -metric and $\{r\} \cup T$ as the terminal set.*

The proof of this theorem is similar to the one given by Jain et al. [6] for the equivalence between α -approximations for maximum fractional Steiner tree packing and for minimum Steiner tree problem and is therefore omitted. The above theorem has many interesting corollaries.

Corollary 1. *If all end-hosts are terminals, i.e., $T = H$, then SBM can be solved optimally in polynomial time.*

A metric is called a tree-metric if it is a G -metric for some tree G . The following theorem which states that tree-metrics are “easy” for the minimum Steiner tree problem may be of independent interest.

Theorem 2. *The minimum Steiner tree problem can be solved optimally in polynomial time on a tree-metric.*

The proof of the above theorem is given in the following section. The theorem above combined with Theorem 1 gives the following corollary.

Corollary 2. *SBM can be solved optimally in polynomial time on a tree network.*

3.1 Minimum Steiner Tree Problem on Tree-Metric

Given a metric, there are methods known to identify if it is a tree-metric and if yes, to construct a weighted binary tree with the points in the metric as leaves that induces the given metric (see [9,1]). Before we present our algorithm for finding a minimum Steiner tree in tree-metric, we describe a transformation called “load minimization” that will be useful in the analysis.

Load Minimization. Consider a tree network $G = (V, E)$ with L as the set of leaves. We are also given a set $T \subseteq L$ of terminals and a root $r \in T$. The tree G is considered rooted and hanging at r . Consider a multicast tree $M = (Q, P)$ that spans $\{r\} \cup T$. The edges of Q are considered directed away from r . For a link $e \in G$, let $G_e \subseteq G$ be the subtree (including e) hanging below link e . Let d_e be the number of paths going into G_e and f_e be the number of paths coming out of G_e . We now describe a transformation called “load minimization” that modifies the multicast tree M to ensure that

- either $d_e = 1$ or $f_e = 0$,
- the tree Q still spans the same set of terminals, and
- the number of paths going through any edge in G does *not* increase.

If $d_e = 0$ then clearly there are no terminals in G_e and $f_e = 0$. Suppose, now, that $d_e > 1$ and $f_e > 0$. There are two cases: (1) $d_e \geq f_e$ or (2) $d_e < f_e$. In the first case, after the transformation, the new values of d_e and f_e would be $d_e - f_e$ and 0 respectively, while in the second case, the new values of d_e and f_e would be 1 and $f_e - d_e + 1$ respectively.

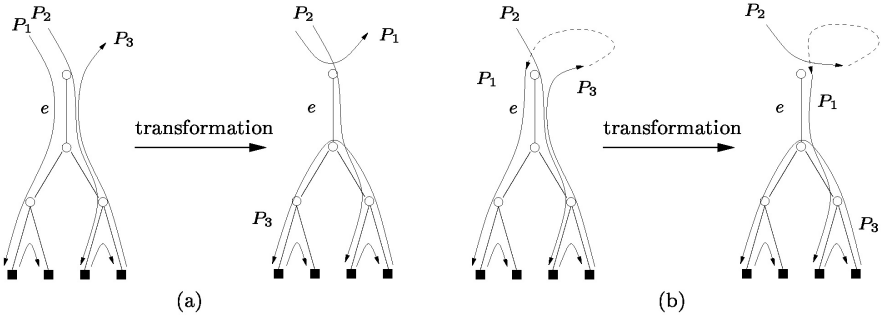


Fig. 2. Applying the transformation “load minimization”: (a) P_2 is *not* an ancestor of P_1 , (b) P_2 is an ancestor of P_1

We first describe the transformation for the case when $d_e = 2$ and $f_e = 1$. There are two paths, say P_1 and P_2 , coming in G_e and one path, say P_3 , going out of G_e . (Refer to Figure 2.) Let f_1, f_2 and f_3 be the edges of Q that correspond to the paths P_1, P_2 and P_3 respectively. There are two cases:

- (a) f_1, f_2 do not have an ancestor-descendant relationship in Q . Note that f_3 is a descendant of one of f_1, f_2 . Suppose it is a descendant of f_2 . We now change

paths P_1, P_3 as follows. The new path P_1 is obtained by gluing together the parts of P_1 and P_3 in $G \setminus G_e$. The new path P_3 is obtained by gluing together the parts of P_1 and P_3 in $G_e \setminus \{e\}$.

- (b) $f_2 \in Q$ is an ancestor of $f_1 \in Q$. f_3 is then a descendant of f_2 and an ancestor of f_1 . In this case, we change paths P_1, P_2, P_3 as follows. The new path P_1 is same as the path P_2 in G_e . The new path P_2 is obtained by gluing the parts of P_2 and P_3 in $G \setminus G_e$. The new path P_3 is obtained by gluing together the parts of P_1 and P_3 in $G_e \setminus \{e\}$.

In both the cases, it is easy to see that this transformation maintains the three required properties. For other values of d_e and f_e , we can apply this transformation by considering two incoming paths and a suitable outgoing path (the edge in Q corresponding to the outgoing path should be a descendant of one of the edges corresponding to the incoming paths). Since by applying the transformation, we reduce both d_e and f_e , we can apply it repeatedly till either $d_e = 1$ or $f_e = 0$.

Algorithm for minimum Steiner tree on tree-metric. Now we describe our dynamic programming based algorithm for finding a minimum Steiner tree on the terminals $T \subseteq L$. Note that the tree induces a metric on the leaves L . Since the internal nodes are not part of the metric-space, they cannot be used as Steiner points. We assume, without loss of generality, that the tree underlying the metric is a binary tree. This can be achieved by adding zero-length links, if necessary. Let M^* be the multicast tree that corresponds to the minimum Steiner tree. Note that the transformation of “load minimization” described above does not increase the number of paths going through any link. Therefore by applying this transformation to M^* , we get another minimum Steiner tree. Thus we can assume that the minimum Steiner tree satisfies that $d_e = 1$ or $f_e = 0$ for each link $e \in G$. Note that if $n = |L|$ is the number of leaves in G , the number of edges in any Steiner tree is at most $n - 1$. Thus, the possible values of (d_e, f_e) are $\mathcal{F} = \{(1, 0), \dots, (1, n - 2), (2, 0), \dots, (n - 1, 0)\}$.

The algorithm finds, for any link $e \in G$ and for any value of $(d_e, f_e) \in \mathcal{F}$, the minimum-length way of routing d_e paths into and f_e paths out of G_e while covering all the terminals (and possibly some non-terminals) in G_e . We denote this quantity by $L_e(d_e, f_e)$. It is easy to find such an information for the leaf links. To find such information for a non-leaf link $e \in G$, we use the information about the child-links e_1 and e_2 of e as follows. We route k_1 paths into T_{e_1} and k_2 paths into T_{e_2} such that $k_1 + k_2 = d_e$. We route k_{12} paths from T_{e_1} to T_{e_2} and k_{21} paths from T_{e_2} to T_{e_1} . We route l_1 paths out of T_{e_1} and l_2 paths out of T_{e_2} such that $l_1 + l_2 = f_e$. Thus, the total number of paths coming in and going out of T_{e_1} is $k_1 + k_{21}$ and $l_1 + k_{12}$ respectively, while the total number of paths coming in and going out of T_{e_2} is $k_2 + k_{12}$ and $l_2 + k_{21}$ respectively. We work with those values of $k_1, k_2, k_{12}, k_{21}, l_1, l_2$ that satisfy $(k_1 + k_{21}, l_1 + k_{12}), (k_2 + k_{12}, l_2 + k_{21}) \in \mathcal{F}$. Since there are only polynomially many choices, we can determine $L_e(d_e, f_e)$ for all values of (d_e, f_e) in polynomial time.

Finally, we modify G so that there is only one link, g , incident at the root. The cost of the minimum Steiner tree is then equal to $\min_i (L_g(i, 0) + l_g \cdot i)$ and can be computed in polynomial time.

4 Unsplittable Bandwidth Maximization

4.1 UBM on Tree Networks

In this section, we give an efficient algorithm to solve the unsplittable bandwidth maximization problem optimally when the input graph G is a tree. We want to find the multicast tree which can route the maximum bandwidth. We solve the decision version of the problem, “Given a bandwidth B , is it possible to construct a multicast tree $M = (Q, P)$ of bandwidth $\lambda_M \geq B$?” and use this to search for the maximum value of bandwidth that can be routed.

Decision Oracle. We replace every end-host at an internal node in the input tree G by a router and attach the end-host as a leaf node to the router with a link of infinite capacity. It is easy to see that this modification does not change the maximum bandwidth that can be routed through the tree and the number of nodes (and links) is at most doubled. With each link e , we associate a label $u_e = \lfloor c_e/B \rfloor$, which represents the maximum number of paths of bandwidth B that can pass through e without exceeding its capacity. For simplicity, we will use the term *path* to mean, a path of bandwidth B , in this section.

For a link e , let G_e be the subtree below. We shall compute two quantities, d_e, f_e called demand and feedback of the link respectively. The demand d_e represents the minimum number of paths that should enter the subtree G_e to satisfy the demands of the terminals in G_e . The feedback f_e represents the maximum number of paths that can emanate from G_e when d_e paths enter G_e . From Section 3.1, it follows that if there is a multicast tree M with $\lambda_M \geq B$, then there is a multicast tree M^* for which $d_e = 1$ or $f_e = 0$ for every link e in G . Since the transformation does not increase the number of paths going through a link, the maximum bandwidth that can be routed does not decrease implying $\lambda_M^* \geq B$.

The oracle does the verification in a bottom-up fashion. If for some link e , its label $u_e < d_e$, the oracle outputs “No” as a multicast tree of bandwidth at least B can not be constructed. If for each link e in the tree, $u_e \geq d_e$, then the oracle outputs “Yes”.

It is easy to compute (d_e, f_e) for a link e incident on a leaf node. Let $v \in H \cup R$ be the leaf node on which e is incident.

$$(d_e, f_e) = \begin{cases} (0, 0) & \text{if } v \in R \\ (0, 0) & \text{if } v \in H - T \text{ and } u_e \leq 2 \\ (1, u_e - 1) & \text{otherwise} \end{cases} \quad (1)$$

In the first two cases, the node can not forward the data to a terminal. Otherwise if a path comes into the end-host (which is clearly its minimum demand), it can replicate the data and send at most $u_e - 1$ paths through the link.

We then compute the values for links incident on internal nodes (routers because of our simplification) in the following manner: Suppose we have computed (d_{e_i}, f_{e_i}) pairs for the child-links $e_1, \dots, e_k$ of a link e . Let $D = \sum_{i=1}^k d_{e_i}$, and let $F = \sum_{i=1}^k f_{e_i}$. Then

$$(d_e, f_e) = \begin{cases} (D - F, 0) & \text{if } D > F \\ (1, \min(u_e - 1, F - D + 1)) & \text{otherwise} \end{cases} \quad (2)$$

The idea is to use only one incoming path to generate all the feedback from child-links with positive feedback and use it to satisfy demands of other links with zero feedback. The remaining feedback can be sent up the tree along the link e or if the feedback generated is not enough to meet the remaining demands then the demand for e is incremented accordingly.

Suppose links $e_1, \dots, e_p$ have a positive feedback and $e_{p+1}, \dots, e_k$ have 0 feedback. Then $d_{e_i} = 1$ for $1 \leq i \leq p$ and $f_{e_i} = 0$ for $p+1 \leq i \leq k$. We initialize $(d_e, f_e) = (1, 0)$. We send one path along e_1 and generate f_1 paths. One of these paths is sent along e_2 to generate a further feedback of f_2 . We proceed in this manner till we generate a feedback of $\sum_{i=1}^p f_{e_i}$ out of which $p-1$ paths are used to satisfy the sub-trees rooted at these p links. Then depending on whether the surplus feedback or sum of demands of remaining nodes is greater, f_e or d_e is incremented.

This idea is captured succinctly in the equation (2). The *min* term ensures that capacity of link is not violated. Note that our computation ensures that $d_e = 1$ when $f_e > 0$, and $f_e = 0$ when $d_e > 1$.

There is a minor technical detail for the case when we compute $(d_e, f_e) = (1, 0)$. If there is no terminal in the subtree G_e , we need to set $d_e = 0$ because we do not want to send a path into it, unless the feedback is at least 1. This information, whether a subtree contains at least one terminal, can be easily propagated “upwards” in the dynamic computation.

It is easy to see that the verification algorithm can be easily modified to construct a multicast tree of bandwidth at least B , when the output is “Yes”.

Lemma 1. *The bottom-up computation of decision oracle terminates in linear time.*

Proof. No link in the graph is considered more than twice in our traversal, once while computing its demand and feedback and once while computing the values for its parent link. It follows that our algorithm is linear.

Maximizing Bandwidth. It is easy to see that when the maximum bandwidth is routed, at least one edge is used to its full capacity. Otherwise, we can increase the bandwidth to utilize the residual capacity. Also note that the maximum number of paths that can pass through a link is $|H| - 1$. So, there are $n \cdot (|H| - 1)$ possible values of the form c_e/k for $e \in E, 1 \leq k < |H|$, for the optimal bandwidth B^* . We now show how B^* can be found in time $O(n \log n)$ plus $O(\log n)$ calls to the decision oracle. Since our decision oracle runs in linear time, we take $O(n \log n)$ time to compute the maximum bandwidth.

Consider the set of links, $E' = \{e \in E \mid e \text{ lies on the unique path from root to a terminal}\}$. Let c_0 be the capacity of the minimum capacity link in E' . Clearly, c_0 is an upper bound on the maximum achievable bandwidth. If we replace each link e with two anti-parallel links of capacity $c_e/2$, we can route a bandwidth of $c_e/2$ by considering the Eulerian tour of this graph. Hence $c_0/2$ is a lower bound on the maximum bandwidth. We also use this idea in Section 4.3, to achieve a 2-approximation for UBM on graphs.

We do a binary search for B^* in the interval $[c_0/2, c_0]$, till the length of the interval becomes smaller than $c_0/2|H|^2$. This can be done by using $O(\log |H|)$ calls to the oracle. Let I be this final interval containing B^* . We now argue that corresponding to each link e , at most one point of the form c_e/k lies in the interval I . Consider the two cases:

- $c_e < c_0/2$: No feasible point corresponding to c_e lies in $[c_0/2, c_0]$ and hence in I .
- $c_e \geq c_0/2$: Consider any value of $k, 1 \leq k < |H|$, such that c_e/k lies in I . For any $k' \neq k$, we have $|\frac{c_e}{k} - \frac{c_e}{k'}| \geq \frac{c_e}{|H|-2} - \frac{c_e}{|H|-1} > \frac{c_e}{|H|^2} \geq \frac{c_0}{2|H|^2}$. Since the length of the interval I is less than $c_0/2|H|^2$, c_e/k' can not lie in I .

So, we have “filtered” out at most n points, one of which is B^* . We now sort these n points and then make another $O(\log n)$ calls to the oracle to find the value of B^* . As mentioned before, this gives a total running time of $O(n \log n)$.

Theorem 3. *There is an $O(n \log n)$ -time algorithm to compute the optimal bandwidth multicast tree when the input graph is a tree on n nodes.*

4.2 Hardness Results for UBM on General Graphs

In this section, we prove the NP-hardness for UBM on graphs by reducing the Hamiltonian cycle problem on 3-regular graphs to it. In fact, our reduction also gives us a lower bound of 2 on the approximation ratio achievable.

The problem of determining if a given 3-regular graph has a Hamiltonian cycle is NP-complete [4]. Given an instance of the Hamiltonian cycle problem on a 3-regular graph, we construct an instance of the bandwidth maximization problem in polynomial time as follows.

Pick an arbitrary vertex s in $G = (V, E)$ and attach two nodes r, t to it. For every other vertex $v_i \in V$, we add three nodes u_i, w_i, z_i and links $(v_i, u_i), (u_i, w_i), (w_i, z_i)$ and (z_i, v_i) . Let $G' = (V', E')$ be the resulting graph (see Figure 3). We designate w_i and r, t as end-hosts and assign a unit capacity to all links in E' .

Lemma 2. *The graph G has a Hamiltonian cycle if and only if we can route a bandwidth of more than $1/2$ to all end hosts in G' from r .*

Proof. Suppose there is a Hamiltonian cycle C in G . For every edge $(v_i, v_j) \in C$, we add an edge from w_i to w_j and the corresponding path $w_i - z_i - v_i - v_j - u_j - w_j$ to the multicast tree. If (s, v_1) and (v_n, s) are the edges incident to s in C then we

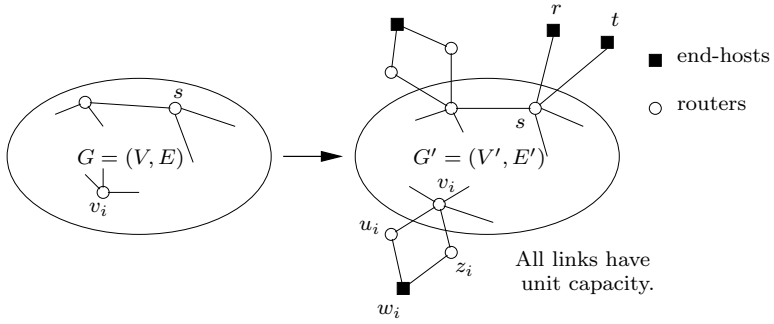


Fig. 3. Polynomial transformation of the Hamiltonian cycle on 3-regular graphs to the bandwidth maximization for multicasting

also add the edges (r, w_1) and (w_n, t) and the corresponding paths $r-s-v_1-u_1-w_1$ and $w_n-z_n-v_n-s-t$ to the multicast tree. Since any link is present on at most one path, a bandwidth of 1 can be routed through this multicast tree.

Suppose now that we can route a bandwidth of greater than $1/2$ through a multicast tree $M = (Q, P)$. As each link has unit capacity, it can belong to at most one path in P . Because at most 2 links are incident to any end-host, its degree in Q is at most 2 and hence Q is in fact a Hamiltonian path on the end-hosts. Now since G is 3-regular, 5 links are incident to any $v_i \in G'$. Out of these, 4 links are used by the two paths incident to w_i . Therefore, the path corresponding to any $(w_j, w_k) \in Q, j \neq i, k \neq i$ cannot contain v_i . Thus it has to travel through the link (v_j, v_k) . Hence an edge (v_j, v_k) must be present in G . These edges together with (s, v_1) and (v_n, s) form a Hamiltonian cycle in G .

Theorem 4. *It is NP-hard to approximate UBM on graphs to better than 2 even when all links have unit capacity and all end-hosts are terminals.*

4.3 A 2-Approximation for UBM on General Graphs

We now give a 2-approximation for UBM on graphs with arbitrary capacities. Our argument is based on a simple upper bound on the maximum bandwidth achievable, as used in Section 4.1.

Lemma 3. *Let h be the largest capacity such that links of capacity at least h connect all the terminals in T to the root r . Then, h is an upper bound on the maximum bandwidth.*

Proof. At least one link of capacity h , or lower, is required to connect all the terminals in T to the root. So any multicast tree has to use at least one such link, and the paths through it can not carry a bandwidth more than h .

The following is a 2-approximation algorithm for UBM.

1. Find the largest capacity h such that links with capacity at least h connect all the terminals to the root.
2. Let G_h be the subgraph induced by links of capacity at least h . Let S be a tree in G_h spanning the root and all terminals.
3. Replace each link e in S with two anti-parallel links of capacity $c_e/2$. Construct an Eulerian tour of this graph and send a bandwidth of $h/2$ through it.

This together with Lemma 3 yields the following theorem.

Theorem 5. *There exists a polynomial time 2-approximation algorithm for UBM.*

5 Conclusions

We consider the problem of computing an overlay tree so as to maximize the throughput achievable for transmitting streaming data. However, we have ignored the latency incurred in sending the data and it would be interesting to design overlay networks which can take both these aspects into account. We have also left unanswered the question of designing multicast trees when we are permitted to use only a specified set of paths between a pair of end-hosts.

Acknowledgments. We would like to thank Ravi Kannan and Arvind Krishnamurthy for suggesting the problem.

References

1. K. Atteson. The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25:251–278, 1999.
2. Y. Chawathe. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, 2000.
3. P. Francis. Yoid: Extending the multicast internet architecture. White paper, 1999. <http://www.aciri.org/yoid/>.
4. M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company, San Francisco, 1979.
5. Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast (keynote address). In *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 1–12. ACM Press, 2000.
6. K. Jain, M. Mahdian, and M. Salavatipour. Packing steiner trees. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, 2003.
7. J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O’Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating System Design and Implementation*, pages 197–212.
8. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS ’01)*, pages 49–60, San Francisco, CA, USA, March 2001.
9. N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

Optimal Distance Labeling for Interval and Circular-Arc Graphs

Cyril Gavoille¹ and Christophe Paul²

¹ LaBRI, Université Bordeaux I, gavoille@labri.fr

² LIRMM, CRNS, paul@lirmm.fr

Abstract. In this paper we design a distance labeling scheme with $\mathcal{O}(\log n)$ bit labels for interval graphs and circular-arc graphs with n vertices. The set of all the labels is constructible in $\mathcal{O}(n)$ time if the interval representation of the graph is given and sorted. As a byproduct we give a new and simpler $\mathcal{O}(n)$ space data-structure computable after $\mathcal{O}(n)$ preprocessing time, and supporting constant worst-case time distance queries for interval and circular-arc graphs. These optimal bounds improve the previous scheme of Katz, Katz, and Peleg (STACS '00) by a $\log n$ factor. To the best of our knowledge, the interval graph family is the first hereditary family having $2^{\Omega(n \log n)}$ unlabeled n -vertex graphs and supporting a $o(\log^2 n)$ bit distance labeling scheme.

Keywords: Data-structure, distance queries, labeling scheme, interval graphs, circular-arc graphs

1 Introduction

Network representation plays an extensive role in the areas of distributed computing and communication networks (including peer-to-peer protocols). The main objective is to store useful information about the network (adjacency, distances, connectivity, etc.) and make it conveniently accessible.

This paper deals with distance representation based on assigning vertex labels [25]. Formally, a *distance labeling scheme* for a graph family $\mathcal{F}$ is a pair $\langle L, f \rangle$ of functions such that $L(v, G)$ is a binary label associated to the vertex v in the graph G , and such that $f(L(x, G), L(y, G))$ returns the distance between the vertices x and y in the graph G , for all x, y of G and every $G \in \mathcal{F}$. The labeling scheme is said an $\ell(n)$ -distance labeling if for every n -vertex graph $G \in \mathcal{F}$, the length of the labels are no more than $\ell(n)$ bits.

A labeling scheme using short labels is clearly a desirable property for a graph, especially in the framework of distributing computing where individual processor element of a network want to communicate with its neighbors but has not enough local memory resources to store all the underlying topology of the network. Schemes providing compact labels play an important role for localized distributed data-structures (see [14] for a survey).

1.1 Related Works

In this framework, Peleg [24] introduced *informative labeling schemes* for graphs and thereby captured a whole set of already known results. Among them implicit representation or adjacency labeling [4,18] whose objective is only to decide adjacency between two vertex labels, compact routing [11,29] whose objective is to provide the first edge on a near-shortest path between the source and the destination, nearest common ancestor and other related functions for trees [2,3,1,19] (used for instance in XML file engine), flow and connectivity [20].

A first set of results about distance labeling can be found in [15]. It is shown, for instance, that general n -vertex graphs enjoy an optimal $\Theta(n)$ -distance labeling, and trees an optimal $\Theta(\log^2 n)$ -distance labeling. It is also proved a lower bound of $\Omega(\sqrt{n})$ on the label length for bounded degree graphs, and of $\Omega(n^{1/3})$ for planar graphs. There are still intriguing gap between upper and lower bounds for these two latter results. The upper bound for planar graphs is $\mathcal{O}(\sqrt{n} \log n)$, coming from a more general result about graphs having small separators. Related works concern distance labeling schemes in dynamic tree networks [22], and approximate distance labeling schemes [12,27,28].

Several efficient schemes have been designed for specific graph families: interval and permutation graphs [21], distance hereditary graphs [13], bounded tree-width graphs (or graphs with bounded vertex-separator), and more generally bounded clique-width graphs [9]. All support an $\mathcal{O}(\log^2 n)$ -distance labeling scheme. Except for the two first families (interval and permutation graphs), these schemes are known to be optimal as these families include trees.

1.2 Our Results

An interval graph is a graph whose vertices are intervals of the real line and whose edges are defined by the intersecting intervals. The interval graph family is hereditary, i.e., a family closed under vertex deletion, and supports a straightforward adjacency labeling with $\mathcal{O}(\log n)$ bit labels. Finding the complexity of distance label length for interval graphs is a difficult problem. The best scheme up to date is due to Katz, Katz, and Peleg [21]. It is based on the particular shape of the separators (that are cliques) and uses $\mathcal{O}(\log^2 n)$ bit labels. The $\Omega(\log^2 n)$ lower bound of [15] does not apply because interval graphs do not contain trees. As shown in [18], information-theoretic lower bound coming from the number of n -vertex graph in the family play an important role for the label length. Kannan, Naor, and Rudich [18] conjecture that any hereditary family containing no more than $2^{k(n)n}$ graphs of n vertices enjoys a $\mathcal{O}(k(n))$ -adjacency distance labeling, $\Omega(k(n))$ being clearly a lower bound. Interval graphs, like trees and several other families cited above (including bounded tree-width, bounded clique-width, bounded genus graphs, etc.), are hereditary and support $\mathcal{O}(\log n)$ bit adjacency labels. However, there are much more interval graphs than all the above mentioned graphs. Indeed, trees, bounded tree-width, bounded clique-width, and bounded genus graphs possess only $2^{\mathcal{O}(n)}$ unlabeled n -vertex graphs. As we will see later there are $2^{\Omega(n \log n)}$ unlabeled interval graphs. Moreover, up

to now, no hereditary graph family is known to support an $o(\log^2 n)$ -distance labeling scheme¹. All these remarks seem to plead in Katz-Katz-Peleg's favor with their $\mathcal{O}(\log^2 n)$ -distance labeling scheme [21].

Surprisingly, we show that the interval graph family enjoys a $5 \log n$ -distance labeling scheme², and that any $\ell(n)$ -distance labeling on this family requires $\ell(n) \geq 3 \log n - \mathcal{O}(\log \log n)$. The lower bound derives from a new estimate of the number of interval graphs. We also show that proper interval graphs, a sub-family of interval graphs, enjoy a $2 \log n$ -distance labeling, and we prove an optimal lower bound of $2 \log n - \mathcal{O}(\log \log n)$.

Moreover, once the labels have been assigned, the distance computation from the labels takes a constant number of additions and comparisons on $\mathcal{O}(\log n)$ bit integers. Even more interesting, the preprocessing time to set all the labels runs optimally in $\mathcal{O}(n)$ time, once the sorted list of intervals of the graph is done. Our scheme extends to circular-arc graphs, a natural generalization of interval graphs, where the same bounds apply.

At this step, it is worth to remark that any $\ell(n)$ -distance labeling scheme on a family $\mathcal{F}$ converts trivially into a non-distributed data-structure for $\mathcal{F}$ of $\mathcal{O}(\ell(n) n / \log n)$ space supporting distance queries within the same time complexity, being assumed that a cell of space can store $\Omega(\log n)$ bits of data. Therefore, our result implies that interval graphs (and circular-arc graphs) have a $\mathcal{O}(n)$ space data-structure, constructible in $\mathcal{O}(n)$ time, supporting constant time distance queries.

This latter formulation implies the result of Chen et al. [6]. However, we highlight that both approaches differ in essence. Their technique consists in building a one-to-one mapping from the vertices of the input graph to the nodes of a rooted tree, say T . Then, distances are computed as follows. Let $l(v)$ be the level of v in T (i.e., the distance from the root), and let $A(i, v)$ be the i -th ancestor of v (i.e., the i -th node on the path from v to the root). The distance d between x and y is computed by: if $l(x) > l(y) + 1$ then $d = l(x) - l(y) - 1 + d_1(z, x)$ where $z = A(l(x) - l(y) - 1, x)$, and where $d_1(z, x)$ is the distance between two nodes whose levels differ by at most 1. The distance d_1 is 1, 2 or 3 and is computed by a case analysis with the interval representation of the involved vertices. Answering query is mainly based on the efficient implementation of *level ancestor queries* on trees (to compute z) given by Berkman and Vishkin [5]. However, this clever scheme cannot be converted into a distributed data-structure as ours for the following reason. As the tree has to support level ancestor queries, it implies that any node, if represented with a local label, can extract any of its ancestors with its level. In particular, x and y can extract from their label their nearest common ancestor and its level, so x and y can compute their distance in T . By

¹ It is not difficult to construct a family of diameter two graphs whose adjacency can be decided with $\mathcal{O}(\log n)$ bit labels (some bipartite graphs for instance), so supporting an $\mathcal{O}(\log n)$ distance labeling scheme. However, "diameter two" is not a hereditary property.

² In this paper all the logarithms are in base two.

the lower bound of [15], this cannot be done in less than $\Omega(\log^2 n)$ bit labels. So, access to a global data-structure is inherent to the Chen et al. [6] approach.

1.3 Outline of the Paper

Let us sketch our technique (due to space limitation proofs have been removed from this extended abstract). The key of our scheme is a reduction to proper interval graphs, namely the family of graphs having an interval representation with only proper intervals, i.e., with no strictly included intervals. We carefully add edges to the input graph (by extending the length of non-proper intervals) to obtain a proper interval graph. Distances in the original graph can be retrieved from the label of two vertices of the proper interval graph and from the original interval (see Section 3).

Therefore, the heart of our scheme is based on an efficient labeling of proper interval graphs, and it is presented in Section 2. First, an integer $\lambda(x)$ is associated to every vertex x with the property that the distance d between x and y is: $d = \lambda(y) - \lambda(x) + \delta(x, y)$, if $\lambda(y) \geq \lambda(x)$, and where $\delta(x, y) = 0$ or 1 . Another key property of λ is that the binary relation δ has the structure of a 2-dimensional poset (thus adjacency is feasible within two linear extensions). Actually, we show how to assign in $\mathcal{O}(n)$ time a number $\pi(x)$ to every x such that $\delta(x, y) = 1$ iff $\pi(x) > \pi(y)$. It gives a $2 \log n$ -distance labeling for proper interval graphs (with constant time decoding), and this is optimal from the lower bound presented in Section 5.

For general interval graphs we construct a $5 \log n$ -distance labeling scheme (see Section 3). In Section 5, we also prove a lower bound of $3 \log n - \mathcal{O}(\log \log n)$. A byproduct of our counting argument used for the lower bound, is a new asymptotic of $2^{2n \log n - o(n \log n)}$ on the number of labeled n -vertex interval graphs, solving an open problem of the 1980's [16].

In Section 4 we show how to reduce distance labeling scheme on circular-arc graphs to interval graphs by “unrolling” twice the input circular-arc graph. Distances can be recovered by doubling the label length.

2 A Scheme for Proper Interval Graphs

For all vertices x, y of a graph G , we denote by $\text{dist}_G(x, y)$ distance between x and y in G .

Proper interval graphs form a sub-family of interval graphs. A layout of an interval graph is *proper* if there is no intervals strictly contained in another one, i.e., there are no intervals $[a, b]$ and $[c, d]$ with $a < c < d < b$. A proper interval graph is an interval graph having a proper layout.

There are several well known characterizations of this sub-family: G is a proper interval graph iff G is an interval graph without any induced $K_{1,3}$ [30]; G is a proper interval graph iff it has an interval representation using unit length interval [26]. The layout of an interval graph, and a proper layout of a proper interval graph can be computed in linear time [8,10,17].

From now on we consider that the input graph $G = (V, E)$, a connected, unweighted and n -vertex proper interval graph, is given by a proper layout $\mathcal{I}$. If the layout is not proper, we use the general scheme described in Section 3. For every vertex x , we denote by $L(x)$ and $R(x)$ respectively the left and the right boundary of the interval $\mathcal{I}(x)$. As done in [6], the intervals are assumed to be sorted according to the left boundary, breaking the tie with increasing right boundary. We will also assume that the $2n$ boundaries are pairwise distinct. If not, it is not difficult to see that in $\mathcal{O}(n)$ time one can scan all the boundaries from $\min_x L(x)$ to $\max_x R(x)$ and compute another layout of G that is still proper, and with sorted and distinct boundaries³.

Let x_0 be the vertex with minimum right boundary. The *base-path* $[x_0, \dots, x_k]$ is the sequence of vertices such that $\forall i > 0$, x_i is the neighbor of x_{i-1} whose $R(x_i)$ is maximum. The *layer partition* $V_1, \dots, V_k$ is defined by:

$$V_i = \{v \mid L(v) < R(x_{i-1})\} \setminus \bigcup_{0 \leq j < i} V_j \quad \text{with } V_0 = \emptyset.$$

Given the sorted list of intervals, the base-path and the layer partition can be

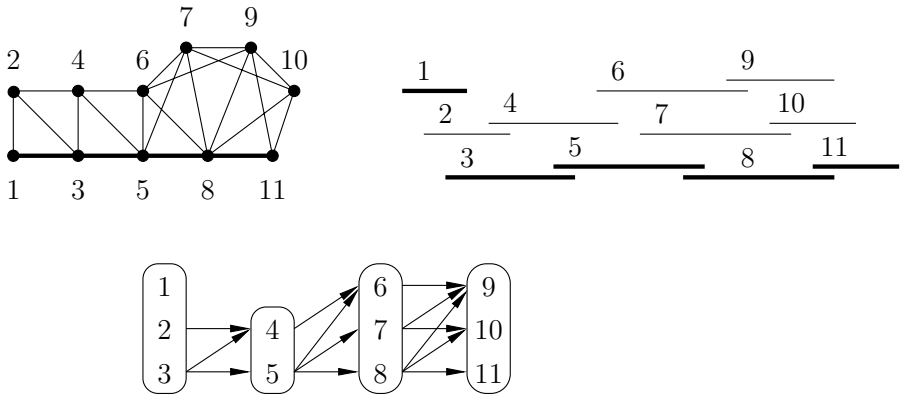


Fig. 1. A proper interval graph with an interval representation and the associated layer partition. The base-path is in bold.

computed in $\mathcal{O}(n)$ time. Let $\lambda(x)$ denote the unique index i such that $x \in V_i$. Let H be the digraph on the vertex set V composed of all the arcs xy such that $\lambda(x) < \lambda(y)$ and $(x, y) \in E$. Note that H is a directed acyclic graph. Let H^t denote the transitive closure of H (since H is acyclic, H^t defines a poset), and let $\text{adj}_{H^t}(x, y)$ be the boolean such that $\text{adj}_{H^t}(x, y) = 1$ iff xy is an arc of H^t .

The scheme we propose is based on the following theorem:

³ Another technical assumption is that all the boundaries of the layout are positive integers bounded by some polynomial of n , so that boundaries can be manipulated in constant time on RAM-word computers. Note that recognizing linear time algorithms produce such layout.

Theorem 1. For all distinct vertices x and y such that $\lambda(x) \leq \lambda(y)$,

$$\text{dist}_G(x, y) = \lambda(y) - \lambda(x) + 1 - \text{adj}_{H^t}(x, y) \quad (1)$$

Therefore to compute the distance between any pair of vertices we have to test whether $\text{adj}_{H^t}(x, y) = 1$. For this reason H^t can be considered as the *graph of errors* associated to the layer partition.

Theorem 2. There exists a linear ordering π of the vertices, constructible in $\mathcal{O}(n)$ time, such that:

$$\text{adj}_{H^t}(x, y) = 1 \text{ iff } \lambda(x) < \lambda(y) \text{ and } \pi(y) < \pi(x) \quad (2)$$

Proof. The ordering π is defined by the following simple algorithm:

π is the pop ordering of a DFS on H using $L(x)$ as a priority rule.

This algorithm can be seen as an elimination ordering of the vertices such that at each step, the sink of the subgraph of H induced by the non-eliminated vertices, that has a minimum left bound, is removed. On the example of Fig. 1, we obtain the following ordering:

vertex x	1	9	6	4	2	10	7	11	8	5	3
$\pi(x)$	1	2	3	4	5	6	7	8	9	10	11

This algorithm can be implemented to run in $\mathcal{O}(n)$ time. First the digraph H can be stored in $\mathcal{O}(n)$ space. The set of vertices are sorted in an array with respect to the left boundaries of their interval. Notice that the layers and also the neighborhood in H of any vertex appear consecutively in this array. So each vertex is associated to its first and last vertex in the array.

The priority rule implies that when a vertex v is popped by the algorithm, any vertex u of the same layer such that $L(u) < L(v)$ has already been popped. It means that when a vertex is at the top of the stack during the DFS, we can find in $\mathcal{O}(1)$ the next vertex to be pushed if the layer of each vertex is also stored and for each layer, the index of the last popped vertex is maintained.

Let us now prove the correctness of Eq. (2).

$\Rightarrow$ It is easy to check that if there is an arc from x to y in H^t , then $\lambda(x) < \lambda(y)$ and $\pi(y) < \pi(x)$.

$\Leftarrow$ So let x and y be two non-adjacent vertices in H^t such that $\lambda(x) < \lambda(y)$ (it implies that $L(x) < L(y)$).

We first show that $\forall z$ such that $xz \in E(H^t)$, then $yz \in E(H^t)$. If $\lambda(x) = \lambda(y)$, yz exists since $L(x) < L(y)$. So assume that $\lambda(x) < \lambda(y)$. If xz exists but not yz , then we can enlighten a $K_{1,3}$ containing x , y and z with a vertex $v \in V_{\lambda(z)}$ that belongs to a shortest x, z -path in G : contradiction, G is a proper interval graph.

By the way at the step y becomes a sink, there remains no vertex z such that $\lambda(y) < \lambda(z)$ and $xz \in E(H^t)$. If x is also a sink, then by the priority rule $\pi(x) < \pi(y)$. If it is not the case, then $\lambda(x) < \lambda(y)$ and there exists a

sink x' such that $xx' \in E(H^t)$ and $\lambda(x') \leq \lambda(y)$. Notice that $L(x') < L(y)$. Otherwise we would have $\lambda(x') = \lambda(y)$ and the existence of a shortest x, x' -path of length $\lambda(x') - \lambda(x)$ (see Theorem 1), would implies the existence of a x, y -path of same length. Therefore xy would be an arc of H^t : contradiction. Since $L(x') < L(y)$, the priority rule implies that x' is popped before y . Recursively applying this argument, we can prove that x will be popped before y by the algorithm and so $\pi(y) < \pi(x)$. $\square$

The *dimension* of a poset P is the minimum number d of linear orderings $\rho_1, \dots, \rho_d$ such that $x <_P y$ iff $x <_{\rho_i} y$ for every i . The next corollary follows immediately from Theorem 2.

Corollary 1. *The graph of errors H^t is a poset of dimension two.*

To each vertex x of G , we assign the label $L(x, G) = \langle \lambda(x), \pi(x) \rangle$. The distance decoder is given by Eq. (1) and (2). Clearly, $\lambda(x), \pi(x) \in \{1, \dots, n\}$, so:

Theorem 3. *The family of n -vertex proper interval graphs enjoys a distance labeling scheme using labels of length $2 \lceil \log n \rceil$, and the distance decoder has constant time complexity. Moreover, given the sorted list of intervals, all the labels can be computed in $\mathcal{O}(n)$ time.*

3 A Scheme for Interval Graphs

In this section, we assume that $G = (V, E)$ is an interval graph, connected and unweighted, and that a layout $\mathcal{I}$ of G is given by a sorted list of intervals. Up to an $\mathcal{O}(n)$ preprocessing time, we assume that interval boundaries are pairwise distinct and taken from the set $\{1, \dots, 2n\}$.

From $\mathcal{I}$ we will build a family of intervals $\mathcal{I}'$ that is the layout of a proper interval graph $G' = (V, E')$ so that the distance in G can be retrieved from the distances in G' plus some additional information. From now $L(x)$ and $R(x)$ will denote the boundaries of $\mathcal{I}(x)$, and $L'(x)$ and $R'(x)$ the boundaries of $\mathcal{I}'(x)$.

Let $x \in V$ such that $\mathcal{I}(x) \subset \mathcal{I}(y)$ for some vertex y , then the *enclosed neighbor* of x , denoted by $N_e(x)$, is the neighbor of x such that $\mathcal{I}(x) \subset \mathcal{I}(N_e(x))$ and such that $R(N_e(x))$ is maximum. As the boundaries of $\mathcal{I}$ are pairwise disjoint, the enclosed neighbor, if it exists, is unique for every vertex. The layout $\mathcal{I}'$ is obtained from $\mathcal{I}$ as follows: if x has an enclosed neighbor then we set $\mathcal{I}'(x) = [L(x), R(N_e(x))]$, and we set $\mathcal{I}'(x) = \mathcal{I}(x)$ otherwise. Let us remark:

1. $L'(x) = L(x)$ and $R'(x) \geq R(x)$;
2. by maximality of its right boundary, $\mathcal{I}(N_e(x)) = \mathcal{I}'(N_e(x))$; and
3. G is a subgraph of G' , and thus $E \subseteq E'$.

Lemma 1. *The layout $\mathcal{I}'$ of G' is proper.*

Note that in the above transformation, the proper layout obtained may have $\mathcal{I}'(x) \subset \mathcal{I}'(y)$ and $R'(x) = R'(y)$. However $\mathcal{I}'$ is a proper layout, so the scheme of Section 2 applies. We say that a vertex x has been *extended* in G' if $R'(x) > R(x)$.

Lemma 2. *If x and y are two vertices such that $R(x) = R'(x) \leq R(y)$. Then $\text{dist}_G(x, y) = \text{dist}_{G'}(x, y)$.*

For any vertex x , define the *maximum neighbor* of x , denoted by $N_m(x)$, as the neighbor of x such that $R(N_m(x))$ is maximum. Like enclosed neighbors, maximum neighbors are not extended in G' . Otherwise we would have $\mathcal{I}(N_m(x)) \subset \mathcal{I}(v)$ for some vertex v and $R(N_m(x)) < R(v)$. Since $\mathcal{I}(x) \cap \mathcal{I}(N_m(x)) \neq \emptyset$, we also have $\mathcal{I}(x) \cap \mathcal{I}(v) \neq \emptyset$. Since $R(N_m(x)) < R(v)$, by definition, we should have $v = N_m(x)$: contradiction.

Theorem 4. *Let x and y be two vertices such that $R(x) \leq R(y)$. Then,*

$$\text{dist}_G(x, y) = \text{dist}_{G'}(N_m(x), y) + 1 - \text{adj}_G(x, y). \quad (3)$$

Eq. (3) directly gives a distance labeling scheme for the family of interval graphs. To each vertex, we have to associate $L(x, G)$:

- $R(x)$ and $L(x)$ the boundaries of $\mathcal{I}(x)$;
- $L(x, G')$ and $L(N_m(x), G')$, the labels of x and $N_m(x)$ in the proper interval graph G' .

The distance decoder is the function described in Eq. (3). It uses the distance decoder of proper interval graph given in the previous section. These labels, composed a priori of 6 integers, can be compacted. Indeed, there is no possible edge in G between any pair of vertices that do not belong to the same or consecutive layers of the layer partition. Thus, we have either $\lambda(N_m(x)) = \lambda(x)$ or $\lambda(N_m(x)) = \lambda(x) + 1$. We do not need to store the pair $\langle \lambda(x), \lambda(N_m(x)) \rangle$. A single bit, say $b(x)$, suffices to recover the second part of this information. Therefore the label of a vertex x is:

$$L(x, G) = \langle L(x), R(x), \lambda(x), \pi(x), b(x), \pi(N_m(x)) \rangle$$

As $1 \leq L(x) < R(x) \leq 2n$, $2 \lceil \log n \rceil + 2$ bits suffice to store the two first integers. The four next fields can be stored with $3 \lceil \log n \rceil + 1$ bits, summing up to $5 \lceil \log n \rceil + 3$. Since all the labels in proper interval graphs can be computed in $\mathcal{O}(n)$, and since G' can be obtained within the same complexity, all the labels of G can be computed in $\mathcal{O}(n)$ time.

Theorem 5. *The family of n -vertex interval graphs enjoys a distance labeling scheme using labels of length $5 \lceil \log n \rceil + 3$, and the distance decoder has constant time complexity. Moreover, given the sorted list of intervals, all the labels can be computed in $\mathcal{O}(n)$ time.*

4 A Scheme for Circular-Arc Graphs

Circular-arc graphs are a natural generalization of interval graphs, in which vertices correspond to arcs of a circle rather than intervals on the real line.

Circular-arc graphs can be recognized in $\mathcal{O}(n^2)$ time [23]. Consider an circular-arc graph G . We can associate to every vertex x of G a range $I(x) = [\theta_r(x), \theta_l(x)]$ of angles where $\theta_r(x)$ and $\theta_l(x)$ are taken clockwise in $[0, 2\pi)$. The range $I(x)$ can be seen as a “cyclic” interval in which $[\theta_r(x), \theta_l(x)]$ for $\theta_r(x) > \theta_l(x)$ stands for $[\theta_r(x), 2\pi) \cup [0, \theta_l(x)]$. The vertices x and y are adjacent if $I(x) \cap I(y) \neq \emptyset$.

Let $x_1, \dots, x_n$ the vertices of G ordered such that $\theta_r(x_i) \leq \theta_r(x_{i+1})$ for every $i \in \{1, \dots, n-1\}$. We associate to G a new intersection graph $\tilde{G}$ with vertex set $V(\tilde{G}) = \bigcup_{1 \leq i \leq n} \{I(x_i^1), I(x_i^2)\}$ where, for every $1 \leq i \leq n$:

$$I(x_i^1) = \begin{cases} [\theta_r(x_i), \theta_l(x_i)] & \text{if } \theta_r(x_i) \leq \theta_l(x_i) \\ [\theta_r(x_i), \theta_l(x_i) + 2\pi] & \text{if } \theta_r(x_i) > \theta_l(x_i) \end{cases}$$

$$I(x_i^2) = \begin{cases} [\theta_r(x_i) + 2\pi, \theta_l(x_i) + 2\pi] & \text{if } \theta_r(x_i) \leq \theta_l(x_i) \\ [\theta_r(x_i) + 2\pi, \theta_l(x_i) + 4\pi] & \text{if } \theta_r(x_i) > \theta_l(x_i) \end{cases}$$

Intuitively, $\tilde{G}$ is obtained by unrolling twice the graph G . To form $\tilde{G}$, we list all the intervals of G according to increasing angle θ_r , starting from the arc of x_1 , and clockwise turning two rounds. So, each vertex x_i in G appears twice in $\tilde{G}$ as x_i^1 and x_i^2 . We check that $\tilde{G}$ is an interval graph.

Lemma 3. *For every $i < j$, $\text{dist}_G(x_i, x_j) = \min \{\text{dist}_{\tilde{G}}(x_i^1, x_j^1), \text{dist}_{\tilde{G}}(x_i^1, x_j^2)\}$.*

Therefore, a distance labeling scheme for interval graphs can be transformed into a scheme for circular-arc graph family by doubling the number of vertices, and the label length.

Theorem 6. *The family of n -vertex circular-arc graphs enjoys a distance labeling scheme using labels of length $\mathcal{O}(\log n)$, and the distance decoder has constant time complexity. Moreover, given the sorted list of intervals, all the labels can be computed in $\mathcal{O}(n)$ time.*

5 Lower Bounds

For any graph family $\mathcal{F}$, let $\mathcal{F}_n$ denote the set of graphs of $\mathcal{F}$ having at most n vertices. Before proving the main results of this section, we need some preliminaries.

An α -graph, for integer $\alpha \geq 1$, is a graph G having a pair of vertices (l, r) , possibly with $l = r$, such that l and r are of eccentricity at most α . Let $H = (G_0, G_1, \dots, G_k)$ be a sequence of α -graphs, and let (l_i, r_i) denote the pair of vertices that defines the α -graph G_i , for $i \in \{0, \dots, k\}$. For each non-null integer sequence $W = (w_1, \dots, w_k)$, we denote by H^W the graph obtained by attaching a path of length w_i between the vertices r_{i-1} and l_i , for every $i \in \{1, \dots, k\}$ (see Fig. 2).

A sub-family $\mathcal{H} \subset \mathcal{F}$ of graphs is α -linkable if $\mathcal{H}$ consists only of α -graphs and if $H^W \in \mathcal{F}$ for every graph sequence H of $\mathcal{H}$ and every non-null integer sequence W .

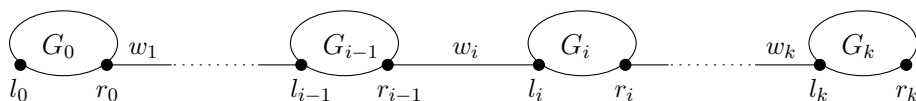


Fig. 2. Linking a sequence of α -graphs.

The following lemma shows a lower bound on the length of the labels used by a distance labeling scheme on any graph family having an α -linkable sub-family. The bound is related to the number of labeled graphs contained in the sub-family. As we will see later, the interval graph family supports a large 1-linkable sub-family (we mean large w.r.t. n), and the proper interval graph family supports large 2-linkable sub-family.

Lemma 4. *Let $\mathcal{F}$ be any graph family, and let $F(N)$ be the number of labeled N -vertex graphs of an α -linkable sub-family of $\mathcal{F}$. Then, every distance labeling scheme on $\mathcal{F}_n$ requires a label of length at least $\frac{1}{N} \log F(N) + \log N - 9$, where $N = \lfloor n/(\alpha \log n) \rfloor$.*

Let us sketch the proof of this lemma. We use a sequence H of $\Theta(\log n)$ α -graphs G_i taken from an arbitrary α -linkable sub-family, each with $N = \Theta(n/\log n)$ vertices, and spaced with paths of length $\Theta(n/\log n)$. Intuitively, the term $\frac{1}{N} \log F(N)$ measures the minimum label length required to decide whether the distance between any two vertices of a same G_i 's is one or two. And the term $\log N$ denotes the minimum label length required to compute the distance between two vertices of consecutive G_i 's. The difficulty is to show that some vertices require both information, observing that one can distribute information on the vertices in a non trivial way. For instance, the two extremity vertices of a path of length w_i does not require $\log w_i$ bit labels, but only $\frac{1}{2} \log w_i$ bits: each extremity can store one half of the binary word representing w_i , and merge their label for a distance query.

Let $I(n)$ be the number of labeled interval graphs with n vertices. At every labeled interval graph G with $n - 1$ vertices, one can associated a new labeled graph G' obtained by attaching a vertex r , labeled n , to all the vertices of G . As the extra vertex is labeled n , all the associated labeled graphs are distinct, and thus their number is at least $I(n - 1)$. The graph G' is an interval 1-graph, choosing $l = r$. Clearly, such interval 1-graphs can be linked to form an interval graph. It turns out that interval graphs have a 1-linkable sub-family with at least $I(n - 1)$ graphs of n vertices.

To get a lower bound on the label length for distance labeling on interval graphs, we can apply Lemma 4 with $I(n)$. However, computing $I(n)$ is an unsolved graph-enumeration problem. Cohen, Komlós and Muller gave in [7] the probability $p(n, m)$ that a labeled n -vertex m -edge random graph is an interval graph under conditions on m . They have computed $p(n, m)$ for $m < 4$, and showed that $p(n, m) = \exp(-32c^6/3)$, where $\lim_{n \rightarrow +\infty} m/n^{5/6} = c$. As the total number of labeled n -vertex m -edge graphs is $\binom{n}{m}$, it follows a formula of

$p(n, m) \cdot \binom{n}{m}$ for the number of labeled interval graphs with $m = \Theta(n^{5/6})$ edges. Unfortunately, using this formula it turns out that $I(n) \geq 2^{\Omega(n^{5/6} \log n)} = 2^{o(n)}$, a too weak lower bound for our needs. The exact number of interval graphs is given up to 30 vertices in [16]. Actually, the generating functions for interval and proper interval graphs (labeled and unlabeled) are known [16], but only an asymptotic of $2^{2n+o(n)}$ for unlabeled proper interval graphs can be estimated from these equations. In conclusion Hanlon [16] left open to know whether the asymptotic on the number of unlabeled interval graphs is $2^{\mathcal{O}(n)}$ or $2^{\Omega(n \log n)}$.

As the number of labeled interval graphs is clearly at least $n! = 2^{(1-o(1))n \log n}$ (just consider a labeled path), the open question of Hanlon is to know whether $I(n) = 2^{(c-o(1))n \log n}$ for some constant $c > 1$. Hereafter we show that $c = 2$, which is optimal.

Theorem 7. *The number $I(n)$ of labeled n -vertex connected interval graphs satisfies $\frac{1}{n} \log I(n) \geq 2 \log n - \log \log n - \mathcal{O}(1)$. It follows that there are $2^{\Omega(n \log n)}$ unlabeled n -vertex interval graphs.*

We have seen that the interval graph family has a 1-linkable sub-family with at least $I(n-1)$ graphs of n vertices. By Theorem 7 and Lemma 4, we have:

Theorem 8. *Any distance labeling scheme on the family of n -vertex interval graphs requires a label of length at least $3 \log n - 4 \log \log n$.*

Using a construction of a 2-linkable sub-family of proper interval graphs with at least $(n-2)!$ graphs of n vertices, one can also show:

Theorem 9. *Any distance labeling scheme on the family of n -vertex proper interval graphs requires a label of length at least $2 \log n - 2 \log \log n - \mathcal{O}(1)$.*

References

1. S. ABITEBOUL, H. KAPLAN, AND T. MILO, *Compact labeling schemes for ancestor queries*, in 12th Symp. on Discrete Algorithms (SODA), 2001, pp. 547–556.
2. S. ALSTRUP, P. BILLE, AND T. RAUHE, *Labeling schemes for small distances in trees*, in 15th Symp. on Discrete Algorithms (SODA), 2003.
3. S. ALSTRUP, C. GAVOILLE, H. KAPLAN, AND T. RAUHE, *Nearest common ancestors: A survey and a new distributed algorithm*, in 14th ACM Symp. on Parallel Algorithms and Architecture (SPAA), Aug. 2002, pp. 258–264.
4. S. ALSTRUP AND T. RAUHE, *Small induced-universal graphs and compact implicit graph representations*, in 43rd IEEE Symp. on Foundations of Computer Science (FOCS), 2002, pp. 53–62.
5. O. BERKMAN AND U. VISHKIN, *Finding level-ancestors in trees*, J. of Computer and System Sciences, 48 (1994), pp. 214–230.
6. D. Z. CHEN, D. LEE, R. SRIDHAR, AND C. N. SEKCHARAN, *Solving the all-pair shortest path query problem on interval and circular-arc graphs*, Networks, 31 (1998), pp. 249–257.
7. J. E. COHEN, J. KOMLÓS, AND T. MUELLER, *The probability of an interval graph, and why it matters*, Proc. of Symposia in Pure Mathematics, 34 (1979), pp. 97–115.

8. D. CORNEIL, H. KIM, S. NATARAJAN, S. OLARIU, AND A. SPRAGUE, *Simple linear time algorithm of unit interval graphs*, Info. Proces. Letters, 55 (1995), pp. 99–104.
9. B. COURCELLE AND R. VANICAT, *Query efficient implementation of graphs of bounded clique width*, Discrete Applied Mathematics, (2001). To appear.
10. C. DE FIGUEIREDO HERRERA, J. MEIDANIS, AND C. PICININ DE MELLO, *A linear-time algorithm for proper interval recognition*, Information Processing Letters, 56 (1995), pp. 179–184.
11. C. GAVOILLE, *Routing in distributed networks: Overview and open problems*, ACM SIGACT News - Distributed Computing Column, 32 (2001), pp. 36–52.
12. C. GAVOILLE, M. KATZ, N. A. KATZ, C. PAUL, AND D. PELEG, *Approximate distance labeling schemes*, in 9th European Symp. on Algorithms (ESA), vol. 2161 of LNCS, Springer, 2001, pp. 476–488.
13. C. GAVOILLE AND C. PAUL, *Distance labeling scheme and split decomposition*, Discrete Mathematics, (2003). To appear.
14. C. GAVOILLE AND D. PELEG, *Compact and localized distributed data structures*, Research Report RR-1261-01, LaBRI, University of Bordeaux, Aug. 2001. To appear in J. of Distributed Computing for the PODC 20-Year Special Issue.
15. C. GAVOILLE, D. PELEG, S. PÉRENNES, AND R. RAZ, *Distance labeling in graphs*, in 12th Symp. on Discrete Algorithms (SODA), 2001, pp. 210–219.
16. P. HANLON, *Counting interval graphs*, Transactions of the American Mathematical Society, 272 (1982), pp. 383–426.
17. P. HELL, J. BANG-JENSEN, AND J. HUANG, *Local tournaments and proper circular arc graphs*, in Algorithms, Int. Symp. SIGAL, vol. 450 of LNCS, 1990, pp. 101–108.
18. S. KANNAN, M. NAOR, AND S. RUDICH, *Implicit representation of graphs*, SIAM J. on Discrete Mathematics, 5 (1992), pp. 596–603.
19. H. KAPLAN, T. MILO, AND R. SHABO, *A comparison of labeling schemes for ancestor queries*, in 14th Symp. on Discrete Algorithms (SODA), 2002.
20. M. KATZ, N. A. KATZ, A. KORMAN, AND D. PELEG, *Labeling schemes for flow and connectivity*, in 13th Symp. on Discrete Algorithms (SODA), 2002, pp. 927–936.
21. M. KATZ, N. A. KATZ, AND D. PELEG, *Distance labeling schemes for well-separated graph classes*, in 17th Symp. on Theoretical Aspects of Computer Science (STACS), vol. 1770 of LNCS, Springer Verlag, 2000, pp. 516–528.
22. A. KORMAN, D. PELEG, AND Y. RODEH, *Labeling schemes for dynamic tree networks*, in 19th Symp. on Theoretical Aspects of Computer Science (STACS), vol. 2285 of LNCS, Springer, 2002, pp. 76–87.
23. R. M. MCCONNELL, *Linear-time recognition of circular-arc graphs*, in 42th IEEE Symp. on Foundations of Computer Science (FOCS), 2001.
24. D. PELEG, *Informative labeling schemes for graphs*, in 25th Int. Symp. on Mathematical Foundations of Computer Science (MFCS), vol. 1893 of LNCS, Springer, 2000, pp. 579–588.
25. ———, *Proximity-preserving labeling schemes*, J. of Graph Theory, 33 (2000).
26. F. ROBERTS, *Indifference graphs*, in Proof Techniques in Graph Theory, Academic Press, 1969, pp. 139–146.
27. M. THORUP, *Compact oracles for reachability and approximate distances in planar digraphs*, in 42th IEEE Symp. on Foundations of Computer Science (FOCS), 2001.
28. M. THORUP AND U. ZWICK, *Approximate distance oracles*, in 33rd ACM Symp. on Theory of Computing (STOC), 2001, pp. 183–192.
29. ———, *Compact routing schemes*, in 13th ACM Symp. on Parallel Algorithms and Architectures (SPAA), 2001, pp. 1–10.
30. G. WEGNER, *Eigenschaften der Neuen homologisch-einfacher Familien im R^n* , PhD thesis, University of Göttingen, 1967.

Improved Approximation of the Stable Marriage Problem

Magnús M. Halldórsson¹, Kazuo Iwama^{2*}, Shuichi Miyazaki³, and
Hiroki Yanagisawa²

¹ Department of Computer Science, University of Iceland
mmh@hi.is

² Graduate School of Informatics, Kyoto University

³ Academic Center for Computing and Media Studies, Kyoto University
{iwama, shuichi, yanagis}@kuis.kyoto-u.ac.jp

Abstract. The stable marriage problem has recently been studied in its general setting, where both ties and incomplete lists are allowed. It is NP-hard to find a stable matching of maximum size, while any stable matching is a maximal matching and thus trivially a factor two approximation.

In this paper, we give the first nontrivial result for approximation of factor less than two. Our algorithm achieves an approximation ratio of $2/(1+L^{-2})$ for instances in which only men have ties of length at most L . When both men and women are allowed to have ties, we show a ratio of $13/7 (< 1.858)$ for the case when ties are of length two. We also improve the lower bound on the approximation ratio to $\frac{21}{19} (> 1.1052)$.

1 Introduction

An instance of the *stable marriage problem* consists of N men, N women and each person's preference list. In a preference list, each person specifies the order (allowing ties) of his/her preference over a subset of the members of the opposite sex. If p writes q on his/her preference list, then we say that q is acceptable to p . A *matching* is a set of pairs of a man and a woman (m, w) such that m is acceptable to w and vice versa. If m and w are matched in a matching M , we write $M(m) = w$ and $M(w) = m$. Given a matching M , a man m and a woman w are said to form a *blocking pair* for M if all the following conditions are met: (i) m and w are not matched together in M but are acceptable to each other. (ii) m is either unmatched in M or prefers w to $M(m)$. (iii) w is either unmatched in M or prefers m to $M(w)$. A matching is called *stable* if it contains no blocking pair.

The problem of finding a stable matching of *maximum size* was recently proved to be NP-hard [14], which also holds for several restricted cases such as the case that all ties occur only in one sex, are of length two and every person's list contains at most one tie [15]. The hardness result has been further

* Supported in part by Scientific Research Grant, Ministry of Japan, 13480081

extended to APX-hardness [8,7]. Since a stable matching is a maximal matching, the sizes of any two stable matchings for an instance differ by a factor at most two. Hence, any stable matching is a 2-approximation; yet, the only non-trivial approximation algorithm is a randomized one for restricted instances [9]. This situation mirrors that of Minimum Maximal Matching [19,20] and Minimum Vertex Cover [10,16], for which, in spite of a long history of research, no approximation of better than a factor of two is known.

Our Contribution. In this paper, we give the first nontrivial upper and lower bounds on the ratio of approximating maximum cardinality solution. On the negative side, it is shown that the problem is hard to approximate within a factor of $\frac{21}{19} (> 1.1052)$. This bound is obtained by showing a non-approximability relation with Minimum Vertex Cover. If the strong conjecture of the $(2 - \epsilon)$ -hardness for the Minimum Vertex Cover holds, then our lower bound will be improved to 1.25.

For the positive side, we give an algorithm called ShiftBrk, which is based on the following simple idea. Suppose, for simplicity, that the length of ties is all the same ($= L$). Then ShiftBrk first breaks all the ties into an arbitrary order and obtain a stable marriage instance *without ties*. Then we “shift” cyclically the order of all the originally tied women in each man’s list simultaneously, creating L different instances. For each of them, we in turn apply the shift operation against the ties of women’s lists, obtaining L^2 instances in total. We finally compute L^2 stable matchings for these L^2 instances in polynomial time, all of which are stable in the original instance [6], and select a largest solution. We prove the following: (i) ShiftBrk achieves an approximation ratio of $2/(1 + L^{-2})$ (1.6 and 1.8 when $L = 2$ and 3, respectively) if the given instance includes ties in only men’s (or women’s) lists. We also give a tight example for this analysis. (ii) It achieves an approximation ratio of $13/7 (< 1.858)$ if $L = 2$. Our conjecture is that ShiftBrk also achieves a factor of less than two for general instances of $L \geq 3$.

Related Work. The stable marriage problem has great practical significance. One of the most famous applications is to assign medical students to hospitals based on the preferences of students over hospitals and vice versa, which are known as NRMP in the US [6], CaRMS in Canada, and SPA in Scotland [12]. Another application is reported in [18], which assigns students to secondary schools in Singapore.

The stable marriage problem was first introduced by Gale and Shapley in 1962 [4]. In its original definition, each preference list must include all members of the opposite sex, and the preference must be in a total order. They proved that every instance admits a stable matching, and gave an $O(N^2)$ -time algorithm to find one, which is called the Gale-Shapley algorithm. Even if ties are allowed in a list, it is easy to find a perfect stable matching using the Gale-Shapley algorithm [6]. If we allow persons to exclude unacceptable partners from the list, the stable matching may no longer be a perfect matching. However, it is well known that all stable matchings for the same instance are of the same size.

Again, it is easy to find a stable matching by the Gale-Shapley algorithm [5]. Hence, the problem of finding a maximum stable matching is trivial in all these three variations, while the situation changes if both ties and incomplete lists are allowed, as mentioned before.

When ties are allowed in the lists, there are two other notions of stability, super-stability and strong stability (in this context, the definition above is sometimes called *weak stability*). In both cases, there can be instances that do not have a stable matching but a polynomial-time algorithm determines its existence and finds one if exists [11]. The book by Gusfield and Irving [6] covers a plenty of results obtained before 80's. In spite of its long history, stable marriage still leaves a lot of open questions which attract researchers [13,1].

2 Notations

Let *SMTI* (Stable Marriage with Ties and Incomplete lists) denote the general stable marriage problem, and *MAX SMTI* be the problem of finding a stable matching of maximum size. *SMI* (Stable Marriage with Incomplete lists) is a restriction of *SMTI*, that do not allow ties in the list.

Throughout this paper, instances contain an equal number N of men and women. We may assume without loss of generality that acceptability is mutual, i.e., that the occurrence of w in m 's preference list implies the occurrence of m in w 's list, and vice versa. A goodness measure of an approximation algorithm T of an optimization problem is defined as usual: the *approximation ratio* of T is the maximum $\max\{T(x)/opt(x), opt(x)/T(x)\}$ over all instances x of size N , where $opt(x)$ ($T(x)$) is the size of the optimal (algorithm's) solution, respectively. A problem is *NP-hard to approximate within $f(N)$* , if the existence of a polynomial-time algorithm with approximation ratio $f(N)$ implies $P=NP$.

If a man (woman) has a partner in a stable matching M , then he/she is said to be *matched* in M , otherwise, is said to be *single*. If m and w are matched in M , we write $M(m) = w$ and $M(w) = m$. If (m, w) is a blocking pair for a matching M , we sometimes say “ (m, w) blocks M ”. If, for example, the preference list of a man m contains w_1, w_2 and w_3 , in this order, we write

$$m : w_1 \ w_2 \ w_3.$$

Two or more persons tied in a list are given in parenthesis, such as

$$m : w_1 \ (w_2 \ w_3).$$

If m strictly prefers w_i to w_j in an instance I , we write “ $w_i \succ w_j$ in m 's list of I .” Let $\hat{I}$ be an *SMTI* instance and let p be a person in $\hat{I}$ whose preference list contains a tie which includes persons $q_1, q_2, \dots, q_k$. In this case, we say that “ $(\dots q_1 \dots q_2 \dots q_k \dots)$ in p 's list of $\hat{I}$.” Let I be an *SMI* instance that can be obtained by breaking all ties in $\hat{I}$, and suppose that the tie $(\dots q_1 \dots q_2 \dots q_k \dots)$ in p 's list of $\hat{I}$ is broken into $q_1 \succ q_2 \succ \dots \succ \dots q_k$ in I . Then we write “ $[\dots q_1 \dots q_2 \dots q_k \dots]$ in p 's list of I .”

3 Inapproximability Results

In this section, we obtain a lower bound on the approximation ratio of MAX SMTI using a reduction from the Minimum Vertex Cover problem (MVC for short). Let $G = (V, E)$ be a graph. A vertex cover C for G is a set of vertices in G such that every edge in E has at least one endpoint in C . The MVC is to find, for a given graph G , a vertex cover with the minimum number of vertices, which is denoted by $VC(G)$. Dinur and Safra [3] gave an improved lower bound of $10\sqrt{5} - 21$ on the approximation ratio of MVC using the following proposition, by setting $p = \frac{3-\sqrt{5}}{2} - \delta$ for arbitrarily small δ . We shall however see that the value $p = 1/3$ is optimal for our purposes.

Proposition 1. [3] *For any $\epsilon > 0$ and $p < \frac{3-\sqrt{5}}{2}$, the following holds: Given a graph $G = (V, E)$, it is NP-hard to distinguish the following two cases:*

- (1) $|VC(G)| \leq (1 - p + \epsilon)|V|$.
- (2) $|VC(G)| > (1 - \max\{p^2, 4p^3 - 3p^4\} - \epsilon)|V|$.

For a MAX SMTI instance $\hat{I}$, let $OPT(\hat{I})$ be a maximum cardinality stable matching and $|OPT(\hat{I})|$ be its size.

Theorem 2. *For any $\epsilon > 0$ and $p < \frac{3-\sqrt{5}}{2}$, the following holds: Given a MAX SMTI instance $\hat{I}$ of size N , it is NP-hard to distinguish the following two cases:*

- (1) $|OPT(\hat{I})| \geq \frac{2+p-\epsilon}{3}N$.
- (2) $|OPT(\hat{I})| < \frac{2+\max\{p^2, 4p^3-3p^4\}+\epsilon}{3}N$.

Proof. Given a graph $G = (V, E)$, we will construct, in polynomial time, an SMTI instance $\hat{I}(G)$ with N men and N women. Let $OPT(\hat{I}(G))$ be a maximum stable matching for $\hat{I}(G)$. Our reduction satisfies the following two conditions: (i) $N = 3|V|$. (ii) $|OPT(\hat{I}(G))| = 3|V| - |VC(G)|$. Then, it is not hard to see that Proposition 1 implies Theorem 2.

Now we show the reduction. For each vertex v_i of G , we construct three men v_i^A , v_i^B and v_i^C , and three women v_i^a , v_i^b and v_i^c . Hence there are $3|V|$ men and $3|V|$ women in total. Suppose that the vertex v_i is adjacent to d vertices $v_{i_1}, v_{i_2}, \dots, v_{i_d}$. Then, preference lists of six people corresponding to v_i are as follows:

$$\begin{array}{ll} v_i^A: v_i^a & v_i^a: v_i^B \ v_{i_1}^C \ \dots \ v_{i_d}^C \ v_i^A \\ v_i^B: (v_i^a \ v_i^b) & v_i^b: v_i^B \ v_i^C \\ v_i^C: v_i^b \ v_{i_1}^a \ \dots \ v_{i_d}^a \ v_i^c & v_i^c: v_i^C \end{array}$$

The order of persons in preference lists of v_i^C and v_i^a are determined as follows: $v_p^a \succ v_q^a$ in v_i^C 's list if and only if $v_p^C \succ v_q^C$ in v_i^a 's list. Clearly, this reduction can be performed in polynomial time. It is not hard to see that condition (i) holds.

We show that condition (ii) holds. Given a vertex cover $VC(G)$ for G , we construct a stable matching M for $\hat{I}(G)$ as follows: For each vertex v_i , if $v_i \in VC(G)$, let $M(v_i^B) = v_i^a$, $M(v_i^C) = v_i^b$, and leave v_i^A and v_i^c single. If $v_i \notin VC(G)$,

let $M(v_i^A) = v_i^a$, $M(v_i^B) = v_i^b$, and $M(v_i^C) = v_i^c$. Fig. 1 shows a part of M corresponding to v_i .

It is straightforward to verify that M is stable in $\hat{I}(G)$. It is easy to see that there is no blocking pair consisting of a man and a woman associated with the same vertex. Suppose there is a blocking pair associated with different vertices v_i and v_j . Then it must be (v_i^C, v_j^a) , and v_i and v_j must be connected in G , so either or both are contained in the optimal vertex cover. By the construction of the matching, this implies that either v_i^C or v_j^a is matched with a person at the top of his/her preference list, which is a contradiction. Hence, there is no blocking pair for M . Observe that $|M| = 2|VC(G)| + 3(|V| - |VC(G)|) = 3|V| - |VC(G)|$. Hence $|OPT(\hat{I}(G))| \geq |M| = 3|V| - |VC(G)|$.

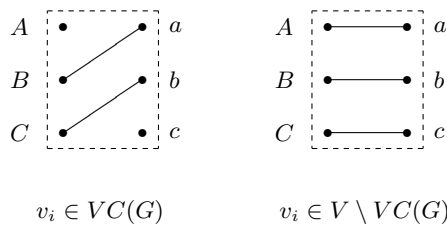


Fig. 1. A part of matching M

Conversely, let M be a maximum stable matching for $\hat{I}(G)$. (We use M instead of $OPT(\hat{I}(G))$ for simplicity.) Consider a vertex $v_i \in V$ and corresponding six persons. Note that v_i^B is matched in M , as otherwise (v_i^B, v_i^b) would block M . We consider two cases according to his partner.

Case (1). $M(v_i^B) = v_i^a$ Then, v_i^b is matched in M , as otherwise (v_i^C, v_i^b) blocks M . Since v_i^B is already matched with v_i^a , $M(v_i^b) = v_i^C$. Then, both v_i^A and v_i^c must be single in M . In this case, we say that “ v_i causes a pattern 1 matching”. Six persons corresponding to a pattern 1 matching is given in Fig. 2.

Case (2). $M(v_i^B) = v_i^b$ Then, v_i^a is matched in M , as otherwise (v_i^A, v_i^a) blocks M . Since v_i^B is already matched with v_i^b , there remain two cases: (a) $M(v_i^a) = v_i^A$ and (b) $M(v_i^a) = v_i^C$ for some j . Similarly, for v_i^C , there are two cases: (c) $M(v_i^C) = v_i^c$ and (d) $M(v_i^C) = v_i^a$ for some j . Hence we have four cases in total. These cases are referred to as patterns 2 through 5 (see Fig. 2). For example, a combination of cases (b) and (c) corresponds to pattern 4.

Lemma 3. *No vertex causes a pattern 3 nor pattern 4 matching.*

Proof. Suppose that a vertex v causes a pattern 3 matching; by mirroring, the same argument holds if we assume that v causes a pattern 4 matching. Then, there is a sequence of vertices $v_{i_1} (= v), v_{i_2}, \dots, v_{i_\ell} (\ell \geq 2)$ such that $M(v_{i_1}^A) = v_{i_1}^a$, $M(v_{i_j}^C) = v_{i_{j+1}}^a$ ($1 \leq j \leq \ell - 1$) and $M(v_{i_\ell}^C) = v_{i_\ell}^c$, namely, v_{i_1} causes a pattern 3

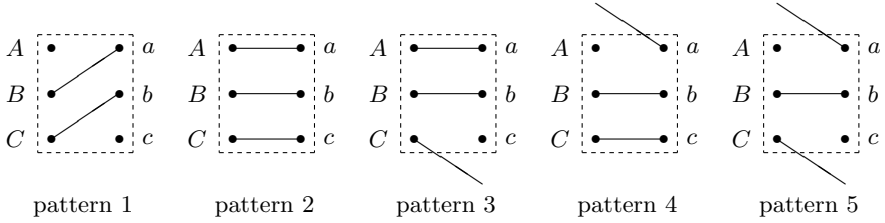


Fig. 2. Five patterns caused by v_i

matching, v_{i_2} through $v_{i_{\ell-1}}$ cause a pattern 5 matching, and v_{i_ℓ} causes a pattern 4 matching.

First, consider the case of $\ell \geq 3$. We show that, for each $2 \leq j \leq \ell - 1$, $v_{i_{j+1}}^a \succ v_{i_{j-1}}^a$ in $v_{i_j}^C$'s list. We will prove this fact by induction.

Since $v_{i_1}^a$ is matched with $v_{i_1}^A$, the man at the tail of her list, $M(v_{i_2}^C)(= v_{i_3}^a) \succ v_{i_1}^a$ in $v_{i_2}^C$'s list; otherwise, $(v_{i_2}^C, v_{i_1}^a)$ blocks M . Hence the statement is true for $j = 2$. Suppose that the statement is true for $j = k$, namely, $v_{i_{k+1}}^a \succ v_{i_{k-1}}^a$ in $v_{i_k}^C$'s list. By the construction of preference lists, $v_{i_{k+1}}^C \succ v_{i_{k-1}}^C$ in $v_{i_k}^a$'s list. Then, if $v_{i_k}^a \succ v_{i_{k+2}}^a$ in $v_{i_{k+1}}^C$'s list, $(v_{i_{k+1}}^C, v_{i_k}^a)$ blocks M . Hence the statement is true for $j = k + 1$.

Now, it turns out that $v_{i_\ell}^a \succ v_{i_{\ell-2}}^a$ in $v_{i_{\ell-1}}^C$'s list, which implies that $v_{i_\ell}^C \succ v_{i_{\ell-2}}^C$ in $v_{i_{\ell-1}}^a$'s list. Then, $(v_{i_\ell}^C, v_{i_{\ell-1}}^a)$ blocks M since $M(v_{i_\ell}^C) = v_{i_\ell}^c$, a contradiction.

It is straightforward to verify that, when $\ell = 2$, $(v_{i_2}^C, v_{i_1}^a)$ blocks M , a contradiction. $\square$

By Lemma 3, each vertex v_i causes a pattern 1, 2 or 5 matching. Construct the subset C of vertices in the following way: If v_i causes a pattern 1 or pattern 5 matching, then let $v_i \in C$, otherwise, let $v_i \notin C$.

We show that C is actually a vertex cover for G . Suppose not. Then, there are two vertices v_i and v_j in $V \setminus C$ such that $(v_i, v_j) \in E$ and both of them cause pattern 2 matching, i.e., $M(v_i^C) = v_i^c$ and $M(v_j^A) = v_j^a$. Then (v_i^C, v_j^a) blocks M , contradicting the stability of M . Hence, C is a vertex cover for G . It is easy to see that $|M| (= |\text{OPT}(\hat{I}(G))|) = 2|C| + 3(|V| - |C|) = 3|V| - |C|$. Thus $|VC(G)| \leq 3|V| - |\text{OPT}(\hat{I}(G))|$. Hence condition (ii) holds. $\square$

The following corollary is immediate from the above theorem by letting $p = \frac{1}{3}$.

Corollary 4. *It is NP-hard to approximate MAX SMTI within any factor smaller than $\frac{21}{19}$.*

Observe that Theorem 2 and Corollary 4 hold for the restricted case where ties occur only in one sex and are of length only two. Furthermore, each preference list is either totally ordered or consists of a single tied pair.

Remark. A long-standing conjecture states that MVC is hard to approximate within a factor of $2 - \epsilon$. We obtain a 1.25 lower bound for MAX SMTI, modulo this conjecture. (Details are omitted, but one can use the same reduction and the fact that MVC has the same approximation difficulty even for the restricted case that $|VC(G)| \geq \frac{|V|}{2}$ [17].)

4 Approximation Algorithm ShiftBrk

In this section, we show our approximation algorithm ShiftBrk, and analyze its performance. Let $\hat{I}$ be an SMTI instance and let I be an SMI instance which is obtained by breaking all ties in $\hat{I}$. Suppose that, in $\hat{I}$, a man m has a tie T of length ℓ consisting of women $w_1, w_2, \dots, w_\ell$. Also, suppose that this tie T is broken into $[w_1 \ w_2 \ \dots \ w_\ell]$ in m 's list of I . We say “shift a tie T in I ” to obtain a new SMI instance I' in which only the tie T is changed to $[w_2 \ \dots \ w_\ell \ w_1]$ and other preference lists are same with I . If I' is the result of shifting all broken ties in men's lists in I , then we write “ $I' = \text{Shift}_m(I)$ ”. Similarly, if I' is the result of shifting all broken ties in women's lists in I , then we write “ $I' = \text{Shift}_w(I)$ ”. Let L be the maximum length of ties in $\hat{I}$.

- Step 1. Break all ties in $\hat{I}$ in an arbitrary order. Let $I_{1,1}$ be the resulting SMI instance.
- Step 2. For each $i = 2, \dots, L$, construct an SMI instance $I_{i,1} = \text{Shift}_m(I_{i-1,1})$.
- Step 3. For each $i = 1, \dots, L$ and for each $j = 2, \dots, L$, construct an SMI instance $I_{i,j} = \text{Shift}_w(I_{i,j-1})$.
- Step 4. For each i and j , find a stable matching $M_{i,j}$ for $I_{i,j}$ using the Gale-Shapley algorithm.
- Step 5. Output a largest matching among all $M_{i,j}$'s.

Since the Gale-Shapley algorithm in Step 4 runs in $O(N^2)$ -time, ShiftBrk runs in polynomial time in N . It is easy to see that all $M_{i,j}$ are stable for $\hat{I}$ (see [6] for example). Hence ShiftBrk outputs a feasible solution.

4.1 Annoying Pairs

Before analyzing the approximation ratio, we will define a useful notion, an annoying pair, which plays an important role in our analysis. Let $\hat{I}$ be an SMTI instance and M_{opt} be a largest stable matching for $\hat{I}$. Let I be an SMI instance obtained by breaking all ties of $\hat{I}$ and M be a stable matching for I . A pair (m, w) is said to be *annoying* for M if they are matched together in M , both are matched to other people in M_{opt} , and both prefer each other to their partners in M_{opt} . That is, (a) $M(m) = w$, (b) m is matched in M_{opt} and $w \succ M_{opt}(m)$ in m 's list of I , and w is matched in M_{opt} and $m \succ M_{opt}(w)$ in w 's list of I .

Lemma 5. *Let (m, w) be an annoying pair for M . Then, one or both of the following holds:*

- (i) $[\dots w \dots M_{opt}(m) \dots]$ in m 's list of I ;
- (ii) $[\dots m \dots M_{opt}(w) \dots]$ in w 's list of I .

Proof. If the strict preferences hold also in $\hat{I}$, i.e. $w \succ M_{opt}(m)$ in m 's list of $\hat{I}$ and $m \succ M_{opt}(w)$ in w 's list of $\hat{I}$, then (m, w) blocks M_{opt} in $\hat{I}$. Thus, either of these preferences in I must have been caused by the breaking of ties in $\hat{I}$. $\square$

Fig. 3 shows a simple example of an annoying pair. (A dotted line means that both endpoints are matched in M_{opt} and a solid line means the same in M . In m_3 's list, w_2 and w_3 are tied in $\hat{I}$ and this tie is broken into $[w_2 \ w_3]$ in I .)

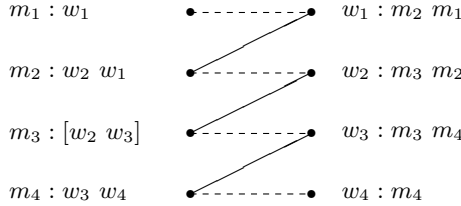


Fig. 3. An annoying pair (m_3, w_2) for M

Lemma 6. *If $|M| < |M_{opt}| - k$ then the number of annoying pairs for M is greater than k .*

Proof. Let M and M' be two stable matchings in an SMTI instance $\hat{I}$. Define a bipartite graph $G_{M, M'}$ as follows. There is a vertex for each person in $\hat{I}$, and an edge between vertices m and w if and only if m and w are matched in M or M' (if they are matched in both, we give two edges between them; hence $G_{M, M'}$ is a multigraph). The degree of each vertex is then at most two, and each connected component of $G_{M, M'}$ is a simple path, a cycle or an isolated vertex.

Consider a connected component C of $G_{M, M_{opt}}$. If C is a cycle (including a cycle of length two), then the number of pairs in M and in M_{opt} included in C is the same. If C is a path, then the number of pairs in M_{opt} could be larger than the number of pairs in M by one. Since $|M| < |M_{opt}| - k$, the number of paths in $G_{M, M_{opt}}$ must be more than k . We show that each path in $G_{M, M_{opt}}$ contains at least one annoying pair for M .

Consider a path $m_1, w_1, m_2, w_2, \dots, m_\ell, w_\ell$, where $w_s = M_{opt}(m_s)$ ($1 \leq s \leq \ell$) and $m_{s+1} = M(w_s)$ ($1 \leq s \leq \ell - 1$). (This path begins with a man and ends with a woman. Other cases can be proved in a similar manner.) Suppose that this path does not contain an annoying pair for M . Since m_1 is single in M , $m_2 \succ m_1$ in w_1 's list of I (otherwise, (m_1, w_1) blocks M). Then, consider the man m_2 . Since we assume that (m_2, w_1) is not an annoying pair, $w_2 \succ w_1$ in m_2 's list of I . We can continue the same argument to show that $m_3 \succ m_2$ in w_2 's list of I and $w_3 \succ w_2$ in m_3 's list of I , and so on. Finally, we have that $w_\ell \succ w_{\ell-1}$ in m_ℓ 's list of I . Since w_ℓ is single in M , (m_ℓ, w_ℓ) blocks M , a contradiction. Hence every path must contain at least one annoying pair and the proof is completed. $\square$

4.2 Performance Analyses

In this section, we consider SMTI instances such that (i) only men have ties and (ii) each tie is of length at most L . Note that we do not restrict the *number* of ties in the list; one man can write more than one ties, as long as each tie is of length at most L . We show that the algorithm ShiftBrk achieves an approximation ratio of $2/(1 + L^{-2})$.

Let $\hat{I}$ be an SMTI instance. We fix a largest stable matching M_{opt} for $\hat{I}$ of cardinality $n = |M_{opt}|$. All preferences in this section are with respect to $\hat{I}$ unless otherwise stated. Since women do not write ties, we have L instances $I_{1,1}, I_{2,1}, \dots, I_{L,1}$ obtained in Step 2 of ShiftBrk, and write them for simplicity as $I_1, I_2, \dots, I_L$. Let $M_1, M_2, \dots, M_L$ be corresponding stable matchings obtained in Step 4 of ShiftBrk.

Let V_{opt} and W_{opt} be the set of all men and women, respectively, who are matched in M_{opt} . Let V_a be a subset of V_{opt} such that each man $m \in V_a$ has a partner in all of $M_1, \dots, M_L$. Let $W_b = \{w | M_{opt}(w) \in V_{opt} \setminus V_a\}$. Note that, by definition, $W_b \subseteq W_{opt}$ and $|V_a| + |W_b| = n$. For each woman w , let $best(w)$ be the man that w prefers the most among $M_1(w), \dots, M_L(w)$; if she is single in each $M_1, \dots, M_L$, then $best(w)$ is not defined.

Lemma 7. *Let w be in W_b . Then $best(w)$ exists and is in V_a , and is preferred by w over $M_{opt}(w)$. That is, $best(w) \in V_a$ and $best(w) \succ M_{opt}(w)$ in w 's list of $\hat{I}$.*

Proof. By the definition of W_b , $M_{opt}(w) \in V_{opt} \setminus V_a$. By the definition of V_a , there is a matching M_i in which $M_{opt}(w)$ is single. Since M_i is a stable matching for $\hat{I}$, w has a partner in M_i and further, that partner $M_i(w)$ is preferred over $M_{opt}(w)$ (as otherwise, $(M_{opt}(w), w)$ blocks M_i). Since w has a partner in M_i , $best(w)$ is defined and differs from $M_{opt}(w)$. By the definition of $best(w)$, w prefers $best(w)$ over $M_{opt}(w)$. That implies that $best(w)$ is matched in M_{opt} , i.e. $best(w) \in V_{opt}$, as otherwise $(best(w), w)$ blocks M_{opt} . Finally, $best(w)$ must be matched in each $M_1, \dots, M_L$, i.e. $best(w) \in V_a$, as otherwise $(best(w), w)$ blocks the M_i for which $best(w)$ is single.

Lemma 8. *Let m be a man and w_1 and w_2 be women, where $m = best(w_1) = best(w_2)$. Then w_1 and w_2 are tied in m 's list of $\hat{I}$.*

Proof. Since $m = best(w_1) = best(w_2)$, there are matchings M_i and M_j such that $m = M_i(w_1) = M_j(w_2)$. First, suppose that $w_1 \succ w_2$ in m 's list. Since $m = M_j(w_2)$, w_1 is not matched with m in M_j . By the definition of $best(w)$, w_1 is either single or matched with a man below m in her list, in the matching M_j . In either case, (m, w_1) blocks M_j in $\hat{I}$, a contradiction. By exchanging the role of w_1 and w_2 , we can show that it is not the case that $w_2 \succ w_1$ in m 's list. Hence w_1 and w_2 must be tied in m 's list of $\hat{I}$. $\square$

By the above lemma, each man can be $best(w)$ for at most L women w because the length of ties is at most L . Let us partition V_a into V_t and $\bar{V}_t$, where

V_t is the set of all men m such that m is $best(w)$ for exactly L women $w \in W_b$ and $\overline{V_t} = V_a \setminus V_t$.

Lemma 9. *There is a matching M_k for which the number of annoying pairs is at most $|M_k| - (|V_t| + \frac{|\overline{V_t}|}{L})$.*

Proof. Consider a man $m \in V_t$. By definition, there are L women $w_1, \dots, w_L$ such that $m = best(w_1) = \dots = best(w_L)$, and all these women are in W_b . By Lemma 8, all these women are tied in m 's list of $\hat{I}$. By Lemma 7, each woman w_i prefers $best(w_i)(= m)$ to $M_{opt}(w_i)$, namely, $m \neq M_{opt}(w_i)$ for any i . This means that none of these women can be $M_{opt}(m)$. For m to form an annoying pair, $M_{opt}(m)$ must be included in m 's tie, due to Lemma 5 (i) (note that the case (ii) of Lemma 5 does not happen because women do not write ties). Hence m cannot form an annoying pair for any of M_1 through M_L .

Next, consider a man $m \in \overline{V_t}$. If $M_{opt}(m)$ is not in the tie of m 's list, m cannot form an annoying pair for any of M_1 through M_L , by the same argument as above. If m writes $M_{opt}(m)$ in a tie, there exists an instance I_i such that $M_{opt}(m)$ lies on the top of the broken tie of m 's list of I_i . This means that m does not constitute an annoying pair for M_i by Lemma 5 (i).

Hence, there is a matching M_k for which at least $|V_t| + \frac{|\overline{V_t}|}{L}$ men, among those matched in M_k , do not form an annoying pair. Hence the number of annoying pairs is at most $|M_k| - (|V_t| + \frac{|\overline{V_t}|}{L})$. $\square$

Lemma 10. $|V_t| + \frac{|\overline{V_t}|}{L} \geq \frac{n}{L^2}$.

Proof. By the definition of V_t , a man in V_t is $best(w)$ for L different women, while a man in $\overline{V_t}$ is $best(w)$ for up to L women. Recall that by Lemma 7, for each woman w in W_b , there is a man in V_a that is $best(w)$. Thus, W_b contains at most $|V_t|L + |\overline{V_t}|(L - 1)$ women. Since $|V_a| + |W_b| = n$, we have that

$$n \leq |V_a| + |V_t|L + |\overline{V_t}|(L - 1) = L|V_a| + |V_t|.$$

Now,

$$\begin{aligned} |V_t| + \frac{|\overline{V_t}|}{L} &= |V_t| + \frac{|V_a| - |V_t|}{L} \\ &= \frac{1}{L}|V_a| + \frac{L-1}{L}|V_t| \\ &\geq \frac{1}{L} \frac{n - |V_t|}{L} + \frac{L-1}{L}|V_t| \\ &= \frac{n}{L^2} + \frac{L^2 - L - 1}{L^2}|V_t| \\ &\geq \frac{n}{L^2}. \end{aligned}$$

The last inequality is due to the fact that $L^2 - L - 1 > 0$ since $L \geq 2$. $\square$

Theorem 11. *The approximation ratio of ShiftBrk is at most $2/(1 + L^{-2})$ for a set of instances where only men have ties of length at most L .*

Proof. By Lemmas 9 and 10, there is a matching M_k for which the number of annoying pairs is at most $|M_k| - n/L^2$. By Lemma 6, $|M_k| \geq n - (|M_k| - \frac{n}{L^2})$, which implies that $|M_k| \geq \frac{L^2+1}{2L^2}n = \frac{1+L^{-2}}{2}n$. $\square$

Remark. The same result holds for men's preference lists being arbitrary partial order. Suppose that each man m 's list is a partial order with width at most L , namely, the maximum number of mutually indifferent women for m is at most L . Then, we can partition its Hasse diagram into L chains [2]. In each “shift”, we give the priority to one of L chains and the resulting total ordered preference list is constructed so that it satisfies the following property: Each member (woman) of the chain with the priority lies top among all women indifferent with her for m in the original partial order. It is not hard to see that the theorem holds for this case.

Also, we can show that when $L = 2$, the performance ratio of ShiftBrk is at most $13/7$, namely better than two, even if we allow women to write ties. However, we need a complicated special case analysis which is lengthy, and hence it is omitted.

4.3 Lower Bounds for ShiftBrk

In this section, we give a tight lower bound for ShiftBrk for instances where only men have ties of length at most L . We show an example for $L = 4$ (although details are omitted, we can construct a worst case example for any L).

$A_1: (a_1 \ b_1 \ c_1 \ d_1)$	$a_1: A_1$
$A_2: (a_2 \ b_2 \ c_2 \ d_2)$	$a_2: A_2$
$B_1: (b_2 \ b_1 \ c_2 \ d_2)$	$b_1: A_1 \ B_1$
$B_2: b_2$	$b_2: A_2 \ B_1 \ B_2 \ C_1 \ D_1$
$C_1: (b_2 \ c_2 \ c_1 \ d_2)$	$c_1: A_1 \ C_1$
$C_2: c_2$	$c_2: A_2 \ C_1 \ C_2 \ D_1 \ B_1$
$D_1: (b_2 \ c_2 \ d_2 \ d_1)$	$d_1: A_1 \ D_1$
$D_2: d_2$	$d_2: A_2 \ D_1 \ D_2 \ B_1 \ C_1$

The largest stable matching for this instance is of size $2L$ (all people are matched horizontally in the above figure). When we apply ShiftBrk to this instance (breaking ties in the same order written above), the algorithm produces $M_1, \dots, M_L$ in Step 3, where $|M_1| = L + 1$ and $|M_2| = |M_3| = \dots = |M_L| = L$.

Let $I_1, \dots, I_L$ be L copies of the above instance and let I_{all} be an instance constructed by putting $I_1, \dots, I_L$ together. Then, in the worst case tie-breaking, ShiftBrk produces L matchings each of which has the size $(L + 1) \cdot 1 + L \cdot (L - 1) = L^2 + 1$, while a largest stable matching for I_{all} is of size $2L^2$. Hence, the approximation ratio of ShiftBrk for I_{all} is $2L^2/(L^2 + 1) = 2/(1 + L^{-2})$. This means that the analysis is tight for any L .

References

1. V. Bansal, A. Agrawal and V. Malhotra, "Stable marriages with multiple partners: efficient search for an optimal solution," In *Proc. ICALP 2003*, to appear.
2. R. P. Dilworth, "A Decomposition Theorem for Partially Ordered Sets," *Ann. Math.* Vol. 51, pp. 161–166, 1950.
3. I. Dinur and S. Safra, "The importance of being biased," In *Proc. of 34th STOC*, pp. 33–42, 2002.
4. D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *Amer. Math. Monthly*, Vol.69, pp. 9–15, 1962.
5. D. Gale and M. Sotomayor, "Some remarks on the stable matching problem," *Discrete Applied Mathematics*, Vol.11, pp. 223–232, 1985.
6. D. Gusfield and R. W. Irving, "The Stable Marriage Problem: Structure and Algorithms," MIT Press, Boston, MA, 1989.
7. M. Halldórsson, R.W. Irving, K. Iwama, D.F. Manlove, S. Miyazaki, Y. Morita, and S. Scott, "Approximability Results for Stable Marriage Problems with Ties," *Theoretical Computer Science*, to appear.
8. M. Halldórsson, K. Iwama, S. Miyazaki, and Y. Morita, "Inapproximability results on stable marriage problems," In *Proc. LATIN 2002*, LNCS 2286, pp. 554–568, 2002.
9. M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa, "Randomized approximation of the stable marriage problem," In *Proc. COCOON 2003*, to appear.
10. E. Halperin, "Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs," In *Proc. 11th SODA*, pp. 329–337, 2000.
11. R. W. Irving, "Stable marriage and indifference," *Discrete Applied Mathematics*, Vol.48, pp. 261–272, 1994.
12. R. W. Irving, "Matching medical students to pairs of hospitals: a new variation on an old theme," In *Proc. ESA 98*, LNCS 1461, pp. 381–392, 1998.
13. R.W. Irving, D.F. Manlove, S. Scott, "Strong Stability in the Hospitals/Residents Problem," In *Proc. STACS 2003*, LNCS 2607, pp. 439–450, 2003.
14. K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita, "Stable marriage with incomplete lists and ties," In *Proc. ICALP 99*, LNCS 1644, pp. 443–452, 1999.
15. D. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita, "Hard variants of stable marriage," *Theoretical Computer Science*, Vol. 276, Issue 1-2, pp. 261–279, 2002.
16. B. Monien and E. Speckenmeyer, "Ramsey numbers and an approximation algorithm for the vertex cover problem," *Acta Inf.*, Vol. 22, pp. 115–123, 1985.
17. G. L. Nemhauser and L. E. Trotter, "Vertex packing: structural properties and algorithms," *Mathematical Programming*, Vol.8, pp. 232–248, 1975.
18. C.P. Teo, J.V. Sethuraman and W.P. Tan, "Gale-Shapley Stable Marriage Problem Revisited: Strategic Issues and Applications," In *Proc. IPCO 99*, pp. 429–438, 1999.
19. M. Yannakakis and F. Gavril, "Edge dominating sets in graphs," *SIAM J. Appl. Math.*, Vol. 38, pp. 364–372, 1980.
20. M. Zito, "Small maximal matchings in random graphs," *Proc. LATIN 2000*, LNCS 1776, pp. 18–27, 2000.

Fast Algorithms for Computing the Smallest k -Enclosing Disc

Sariel Har-Peled* and Soham Mazumdar

Department of Computer Science, University of Illinois
1304 West Springfield Ave, Urbana, IL 61801, USA
{sariel, smazumda}@uiuc.edu

Abstract. We consider the problem of finding, for a given n point set P in the plane and an integer $k \leq n$, the smallest circle enclosing at least k points of P . We present a randomized algorithm that computes in $O(nk)$ expected time such a circle, improving over all previously known algorithms. Since this problem is believed to require $\Omega(nk)$ time, we present a linear time δ -approximation algorithm that outputs a circle that contains at least k points of P , and of radius less than $(1 + \delta)r_{opt}(P, k)$, where $r_{opt}(P, k)$ is the radius of the minimal disk containing at least k points of P . The expected running time of this approximation algorithm is $O(n + n \cdot \min(\frac{1}{k\delta^3} \log^2 \frac{1}{\delta}, k))$.

1 Introduction

Shape fitting, a fundamental problem in computational geometry, computer vision, machine learning, data mining, and many other areas, is concerned with finding the best shape which “fits” a given input. This problem has attracted a lot of research both for the exact and approximation versions, see [3,11] and references therein.

Furthermore, solving such problems in the real world is quite challenging, as noise in the input is omnipresent and one has to assume that some of the input points are noise, and as such should be ignored. See [5,7,14] for some recent relevant results. Unfortunately, under such noisy conditions, the shape fitting problem becomes notably harder.

An important class of shape fitting problems involve finding an optimal k point subsets from a set of n points based on some optimizing criteria. The optimizing criteria could be the smallest convex hull volume, the smallest enclosing ball, the smallest enclosing box, the smallest diameter amongst others [7,2].

An interesting problem of this class is that of computing the smallest disc which contains k points from a given set of n points in a plane. The initial approaches to solving this problem involved first constructing the order- k Voronoi diagram, followed by a search in all or some of the Voronoi cells. The best known algorithm to compute the order- k Voronoi diagram has time complexity

* Work on this paper was partially supported by a NSF CAREER award CCR-0132901.

$O(nk + n \log^3 n)$. See Agarwal *et al.* [16]. Eppstein and Erickson [7] observed that instead of Voronoi cells, one can work with some $O(k)$ nearest neighbors to each point. The resulting algorithm had a running time of $O(n \log n + nk \log k)$ and space complexity $O(kn + k^2 \log k)$. Using the technique of parametric search, Efrat *et al.* [10] solved the problem in time $O(nk \log^2 n)$ and space $O(nk)$. Finally Matoušek [13] by using a suitable randomized search gave a very simple algorithm which used $O(nk)$ space and had $O(n \log n + nk)$ expected running time.

We revisit this classical problem, and present an algorithm with $O(nk)$ expected running time, that uses $O(n + k^2)$ space. The main reason why this result is interesting is because it beats the lower bound of $\Omega(n \log n)$ on the running time for small k , which follows from element uniqueness in the comparison model. We achieve this by using randomization and the floor function (interestingly enough, this is also the computation model used by Matoušek [13]). Despite this somewhat small gain, removing the extra $\log n$ factor from the running time was a non-trivial undertaking, requiring some new ideas.

The key ingredient in our algorithm is a new *linear* time 2-approximation algorithm, Section 3. This significantly improves over the previous best result of Matoušek [13] that runs in $O(n \log n)$ time. Using our algorithm and the later half of the algorithm of Matoušek (with some minor modifications), we get the new improved exact algorithm. Finally, in Section 4, we observe that from the 2-approximation algorithm one can get a δ -approximation algorithm which is *linear* in n and has polynomial dependence on $1/\delta$.

2 Preliminaries

For a point $p = (x, y)$ in $\mathbb{R}^2$, define $G_r(p)$ to be the point $(\lfloor x/r \rfloor r, \lfloor y/r \rfloor r)$. We call r the *width* of G_r . Observe that G_r partitions the whole space into square regions, which we call grid cells. Formally, for any $i, j \in \mathbb{Z}$, the intersection of the halfplanes $x \geq ri, x < r(i+1), y \geq rj$ and $y < r(j+1)$ is said to be a grid cell. Further, we call a block of 3×3 contiguous grid cells as a *grid cluster*.

For a point set P , and parameter r , the partition of P into subsets by the grid G_r , is denoted by $G_r(P)$. More formally, two points $p, q \in P$ belong to the same set in the partition $G_r(P)$, if both points are being mapped to the same grid point or equivalently belong to the same grid cell. With a slight abuse of notation, we call the partitions as grid cells of $G_r P$.

Let $\text{gd}_P(r)$ denote the maximum number of points of P mapped to a single point by the mapping G_r . Define $\text{depth}(P, r)$ to be the maximum number of points of P that a disc of radius r can contain.

The above notation is originally from Matoušek [13]. Using simple packing arguments we can prove the following results [13]

Lemma 1. $\text{depth}(P, Ar) \leq (A + 1)^2 \text{depth}(P, r)$

Lemma 2. $\text{gd}_P(r) \leq \text{depth}(P, r) = O(\text{gd}_P(r))$.

Lemma 3. *Any disk of radius r can be covered by some grid cluster in G_r .*

We further require the following lemma for our algorithm

Lemma 4. *Let $S_1, S_2, \dots, S_t$ be t finite subsets of $\mathbb{R}^2$ and $B_1, \dots, B_t$ be the respective axis parallel bounding squares. Let $r_1, r_2, \dots, r_t$ be the width of $B_1, \dots, B_t$ respectively. If $B_1, \dots, B_t$ are disjoint and $k \leq |S_i| = O(k)$, then $k \leq \text{depth}(S_1 \cup S_2 \dots \cup S_t, r_{\min}) = O(k)$, where $r_{\min} = \min(r_1, r_2, \dots, r_t)$.*

Proof. Let $S = S_1 \cup S_2 \cup \dots \cup S_t$. It is clear that $\text{depth}(S, r_{\min}) \geq k$ since if $r_{\min} = r_p$, then $\text{depth}(S, r_{\min}) \geq \text{depth}(S_p, r_p) \geq k$. Now consider an arbitrary circle C of radius $r_{\min}$, centered at say a point c . Let B be the axis parallel square of side length $4r_{\min}$ centered as c . Any square of side length greater than $r_{\min}$ which intersects C must have an intersection of area larger than $r_{\min}^2$ with B . This implies that the number of disjoint squares, of side length greater than $r_{\min}$, which can have a non empty intersection with C is at most 16. In particular this means that at-most 16 of the sets $S_1, S_2, \dots, S_t$ can have a non-empty intersection with C . The desired result follows.

Note that the analysis of Lemma 4 is quite loose. The constant 16 can be brought down to 10 with a little more work.

Remark 1. It is important to note that the requirement in Lemma 4, of all sets having at least k points, can be relaxed as follows: It is sufficient that the set S_i with the smallest bounding square B_i must contain at least k points. In particular, the other sets may have fewer than k points and the result would still be valid.

Definition 1 (Gradation). *Given a set P of n points, a sampling sequence $(S_1, \dots, S_m)$ of P is a sequence of sets, such that (i) $S_1 = P$, (ii) S_i is formed by picking each point of S_{i-1} into S_i with probability half, and (iii) $|S_m| \leq n/\log n$, and $|S_{m-1}| > n/\log n$. The sequence $(S_m, S_{m-1}, \dots, S_1)$ is a gradation of P .*

Lemma 5. *Given P , a sampling sequence can be computed in expected linear time.*

Proof. Observe that the sampling time is $O(\sum_{i=1}^m |S_i|)$, where m is the length of the sequence. Note, that

$$\mathbf{E}[|S_i|] = \mathbf{E}\left[\mathbf{E}\left[|S_i| \mid |S_{i-1}|\right]\right] = \mathbf{E}\left[\frac{|S_{i-1}|}{2}\right] = \frac{n}{2^{i-1}}.$$

Thus, $O(\sum_{i=1}^m |S_i|) = O(n)$.

3 Algorithm for Approximation Ratio 2

3.1 The Heavy Case ($k = \Omega(n)$)

Assume that $k = \Omega(n)$, and let $\varepsilon = k/n$. We compute an optimal sized ε -net for the set system $(P, \mathcal{R})$, where $\mathcal{R}$ is the set of all intersections of P with

circular discs in the plane. The VC dimension of this space is four, and hence the computation can be done in $O(n)$ time using deterministic construction of ε -nets. [4]. Note that the size of the computed set is $O(1)$. Let S be the ε -net computed. Let $D_{opt}(P, k)$ be a disc of minimal radius which contains k points of P .

From the definition of ε -nets, it follows that $\exists z \in S$, such that $z \in D_{opt}(P, k)$. Now notice that for an arbitrary $s \in S$, if s' is the $(k-1)$ th closest point to s to P then if $s \in D_{opt}(P, k)$, then $\text{dist}(s, s') \leq 2r_{opt}(P, k)$. This follows because at least $(k-1)$ points in $P \setminus \{s\}$ are in $D_{opt}(P, k)$ and hence they are at a distance $\leq 2r_{opt}(P, k)$ from s . For each point in S , we compute it's distance from the $(k-1)$ th closest point to it. Let r be the smallest of these $|S|$ distances. From the above argument, it follows that $r_{opt}(P, k) \leq r \leq 2r_{opt}(P, k)$. The selection of the $(k-1)$ th closest point can be done deterministically in linear time, by using deterministic median selection [6].

Also note that the radius computed in this step, is one of $O(n)$ possible pairwise distances between a point in P and its k -th closest neighbor. We will make use of this fact in our subsequent discussion

Lemma 6. *Given a set P of n points in the plane, and parameter $k = \Omega(n)$, one can compute in $O(n)$ deterministic time, a disc D that contains k points of P , and $\text{radius}(D) \leq 2r_{opt}(P, k)$.*

We call the algorithm described above as APPROXHEAVY. Note that the algorithm can be considerably simplified by using random sampling to compute the ε -net instead of the deterministic construction.

Using the above algorithm, together with Lemma 4, we can get a moderately efficient algorithm for the case when $k = o(n)$. The idea is to use the algorithm from Lemma 6 to divide the set P into subsets such that the axis parallel bounding squares of the subsets are disjoint, each subset contains $O(k)$ points and further at-least one of the subsets with smallest axis parallel bounding square contains at least k points. If r_s is the width of the smallest of the bounding squares, then clearly $k \leq \text{depth}(P, r_s) = O(k)$ from Lemma 4 and Remark 1

The computation of r_s is done using a divide and conquer strategy. For $n > 20k$, set $k' = n/20$. Using the algorithm for Lemma 6, compute a radius r' , such that $k' \leq \text{gd}_P(r') = O(k')$. Next compute, in linear time, the grid $G_{r'}(P)$. For each grid cell in $G_{r'}(P)$ containing more than k points, apply the algorithm recursively. The output, r_s is the width of the smallest grid cell constructed over all the recursive calls. For $n \leq 20k$, the algorithm simply returns the width of the axis parallel bounding square of P .

See Figure 1 for the divide and conquer algorithm. Observe that the choice of $k' = n/20$ is not arbitrary. We would like r' to be such that $\text{gd}_P(r') \leq n/2$. Since Lemma 6 gives a factor-2 approximation, using Lemma 1 and Lemma 2 we see that the desired condition is indeed satisfied by our choice of k' .

Once we have r_s , we compute $G_{r_s}(P)$. From Lemma 4 we know that each grid cell has $O(k)$ points. Also any circle of radius r_s is entirely contained in some grid

```

APPROXDC(P,k)
  Output:  $r_s$ 
  begin
    if  $|P| \leq 20k$ 
      return width of axis parallel bounding square of P
     $k' \leftarrow |P|/20$ 
    Compute  $r'$  using algorithm from Lemma 6 on  $(P, k')$ 
     $G \leftarrow G_{r'}$ 
    for every grid cell  $c \in G$  with  $|c \cap P| > k$  do
       $r_c \leftarrow \text{APPROXDC}(c \cap P, k)$ 
    return minimum among all  $r_c$  computed in previous step.
  end

```

Fig. 1. The Divide and Conquer Algorithm

cluster. Using the algorithm from Lemma 6 we compute the 2-approximation to the smallest k enclosing circle in each cluster which contains more than k points and then finally output the circle of smallest radius amongst the circles computed for the different clusters.

The correctness of the algorithm is immediate. The running time can be bounded as follows. From Lemma 6, each execution of the *divide* step takes a time which is linear in the number of points in the cell being split. Also the depth of the recursion tree is $O(\log(n/k))$. Thus the time to compute r_s is $O(n \log(n/k))$. Once we have r_s , the final step, to compute a 2-approximation to r_{opt} , takes a further $O(n)$ time. Hence the overall running time of the algorithm is $O(n \log(n/k))$. This result in itself is a slight improvement over the $O(n \log n)$ time algorithm for the same purpose in Matoušek [13].

Lemma 7. *Given a set P of n points in the plane, and parameter k , one can compute in $O(n \log(n/k))$ deterministic time, a disc D that contains k points of P , and $\text{radius}(D) \leq 2r_{opt}(P, k)$.*

Remark 2. For a point set P of n points, the radius returned by the algorithm of Lemma 7 is a distance between some pair of points of P . As such, a grid computed from the distance returned in the previous lemma is one of $O(n^2)$ possible grids.

3.2 General Algorithm

As done in the previous section, we construct a grid which partitions the points into small ($O(k)$ sized) groups. The key idea behind speeding up the grid computation is to construct the appropriate grid over several rounds. Specifically, we start with a small set of points as seed and construct a suitable grid for this subset. Next, we incrementally insert the remaining points, while adjusting the grid width appropriately at each step.

Let $\mathcal{P} = (P_1, \dots, P_m)$ be a *gradation* of P (see Definition 1), where $|P_1| \geq \max(k, n/\log n)$ (i.e. if $k \geq n/\log(n)$ we start from the first set in $\mathcal{P}$ that has more than k elements). The sequence $\mathcal{P}$ can be computed in expected linear time as shown in Lemma 5. Now using the algorithm of Lemma 7, we obtain a length r_1 such that $\text{gd}_{r_1}(P_1) \leq \alpha k$ where α is a suitable constant independent of n and k . The value of α will be established later. The set P_1 is the seed subset mentioned earlier. Observe that it takes $O(|P_1| \log(|P_1|/k)) = O(n)$ time to perform this step.

```

GROW( $P_i, P_{i-1}, r_{i-1}, k$ )
  Output:  $r_i$ 
  begin
     $G_i \leftarrow G_{r_{i-1}}(P_i)$ 
    for every grid cluster  $c \in G_i$  with  $|c \cap P_i| \geq k$  do
       $P_c \leftarrow c \cap P_i$ 
      Compute a distance  $r_c$  such that  $r_{\text{opt}}(P_c, k) \leq r_c \leq 2r_{\text{opt}}(P_c, k)$ ,
        using the algorithm of Lemma 7 on  $P_c$ .

    return minimum  $r_c$  over all clusters.
  end

```

Fig. 2. Algorithm for the i th round

The remaining algorithm works in m rounds, where m is the length of the sequence $\mathcal{P}$. Note that from the sampling sequence construction given in Lemma 5, it is clear that $E[m] = O(\log \log n)$. At the end of the i th round, we have a distance r_i such that $\text{gd}_{r_i}(P_i) \leq \alpha k$, and there exists a grid cluster in G_{r_i} containing more than k points of P_i and $r_{\text{opt}}(P_i, k) \leq r_i$.

At the i th round, we first construct a grid for points in P_i using r_{i-1} as grid width. We know that there is no grid cell containing more than αk points of P_{i-1} . Intuitively, we expect that the points in P_i would not cause any cell to get too heavy, thus allowing us to use the linear time algorithm of Lemma 6 on most grid clusters. The algorithm used in the i th round is more concisely stated in Figure 2. At the end of the m rounds we have r_m , which is a 2-approximation to the radius of the optimal k enclosing disc of $P_m = P$. The overall algorithm is summarized in Figure 3

Analysis

Lemma 8. *For $i = 1, \dots, m$, we have $r_{\text{opt}}(P_i, k) \leq r_i \leq 2r_{\text{opt}}(P_i, k)$. Furthermore, the heaviest cell in $G_{r_i}(P_i)$ contains at most αk points, where $\alpha = 5$.*

Proof. Consider the optimal disk D_i that realizes $r_{\text{opt}}(P_i, k)$. Observe that there is a cluster c of $G_{r_{i-1}}$ that contains D_i , as $r_{i-1} \geq r_i$. Thus, when GROW handles the cluster c , we have $D_i \cap P_i \subseteq c$. The first part of the lemma then follows from the correctness of the algorithm in Lemma 7.

As for the second part, observe that any grid cell of width r_i can be covered with 5 disks of radius $r_i/2$. It follows that the grid cell of $r_{opt}(P_i, k)$ contains at most $5k$ points.

```

LINEARAPPROX(P,k)
  Output:  $r_{2approx}$ 
begin
  Compute a gradation  $\{P_1, \dots, P_m\}$  of  $P$  as in Lemma 5
   $r_1 \leftarrow \text{APPROXDC}(P_1, k)$ 
  for  $j$  going from 2 to  $m$  do
     $r_j \leftarrow \text{GROW}(P_j, P_{j-1}, r_{j-1}, k)$ 
    for every grid cluster  $c \in G_{r_m}$  with  $|c \cap P| \geq k$  do
       $r_c \leftarrow \text{APPROXHEAVY}(c \cap P, k)$ 
    return minimum  $r_c$  computed over all clusters
end

```

Fig. 3. 2-Approximation Algorithm

Definition 2. For a point set P , and a parameters k and r , the excess of $G_r(P)$ is

$$\mathcal{E}(P, k, G_r) = \sum_{c \in \text{Cells of } G_r} \left\lfloor \frac{|c \cap P|}{10\alpha k} \right\rfloor.$$

Remark 3. The quantity $20\alpha k \cdot \mathcal{E}(P, k, G_r)$ is an upper bound on the number of points of P in a heavy cell of P , where a cell of $G_r(P)$ is *heavy* if it contains more than $10\alpha k$ points. The constant α can be taken to be 5 as in Lemma 8.

Lemma 9. For any positive real t , the probability that $G_{r_{i-1}}(P_i)$ has an excess $\mathcal{E}(P_i, k, G_{r_{i-1}}) = M \geq t + 2 \lceil \log(n) \rceil$, is at most 2^{-t} .

Proof. Let $\mathfrak{G}$ be the set of $O(n^2)$ possible grids that might be considered by the algorithm (see Remark 2), and fix a grid $G_r \in \mathfrak{G}$ with excess M .

Let $U = \left\{ P_i \cap c \mid c \in G_r, |P_i \cap c| > 10\alpha k \right\}$ be all the heavy cells in $G_r(P_i)$. Furthermore, let $V = \bigcup_{X \in U} \psi(X, 10\alpha k)$, where $\psi(X, \nu)$ denotes an arbitrary partition of the set X into as many disjoint subsets as possible, such that each subset contains at least ν elements.

It is clear that $|V| = \mathcal{E}(P_i, k, G_r)$. From the chernoff inequality, for any $S \in V$,

$$\Pr[|S \cap P_{i-1}| \leq \alpha k] < \exp\left(-\frac{5\alpha k(1 - 1/5)^2}{2}\right) < \frac{1}{2}$$

Furthermore, $G_r = G_{r_{i-1}}$ only if each cell in $G_r(P_{i-1})$ contains at most αk points. Thus we have

$$\begin{aligned} \Pr[(G_{r_{i-1}} = G_r) \cap (\mathcal{E}(P_i, k, G_r) = M)] &\leq \Pr[G_{r_{i-1}} = G_r \mid \mathcal{E}(P_i, k, G_r) = M] \\ &\leq \prod_{S \in V} \Pr[|S \cap P_{i-1}| \leq \alpha k] \\ &\leq \frac{1}{2^{|V|}} = \frac{1}{2^M}. \end{aligned}$$

There are $\binom{n}{2}$ different grids in $\mathfrak{G}$, and thus we have

$$\begin{aligned} \Pr[\mathcal{E}(P_i, k, G_{r_{i-1}}) = M] &= \Pr\left[\bigcup_{G_r \in \mathfrak{G}} (G_r = G_{r_{i-1}}) \cap \mathcal{E}(P_i, k, G_r) = M\right] \\ &\leq \binom{n}{2} \frac{1}{2^M} \leq \frac{1}{2^t} \end{aligned}$$

Lemma 10. *The probability that $G_{r_{i-1}}(P_i)$ has excess larger than t , is at most 2^{-t} , for $k \geq 4 \log n$.*

Proof. We use the same technique as in Lemma 9. By the Chernoff inequality, the probability that any $10\alpha k$ size subset of P_i would contain at most αk points of P_{i-1} , is less than

$$\leq \exp\left(-5\alpha k \cdot \frac{16}{25} \cdot \frac{1}{2}\right) \leq \exp(-\alpha k) \leq \frac{1}{n^4}.$$

In particular, arguing as in Lemma 9, the probability that $\mathcal{E}(P_i, k, G_{r_{i-1}})$ exceeds t , is smaller than $\binom{n}{2}/n^{4t} \leq 2^{-t}$.

Thus, if $k \geq 4 \log n$, the expected running time of the i th step is at most

$$\begin{aligned} O\left(\sum_{c \in G_{r_{i-1}}} |c \cap P_i| \log \frac{|c \cap P_i|}{k}\right) &= O\left(|P_i| + \sum_{t=1}^{\infty} \frac{tk \log t}{2^t}\right) \\ &= O(|P_i| + k) = O(|P_i|), \end{aligned}$$

For the light case, where $k < 4 \log n$, we have that the expected running time of the i th step is at most

$$\begin{aligned} O\left(\sum_{c \in G_{r_{i-1}}} |c \cap P_i| \log \frac{|c \cap P_i|}{k}\right) &= O\left(|P_i| + k \log n \times \log n + \sum_{t=1+2\lceil \log n \rceil}^{\infty} \frac{tk \log t}{2^t}\right) \\ &= O(|P_i| + k \log^2 n) = O(|P_i|) \end{aligned}$$

Thus, the total expected running time is $O(\sum_i |P_i|) = O(n)$, by the analysis of Lemma 5.

To compute a *factor 2* approximation, consider the grid $G_{r_m}(P)$. Each grid cell contains at-most αk points hence each grid cluster contains at most $9\alpha k$ points which is still $O(k)$. Also the smallest k enclosing disc is contained in some grid cluster. In each cluster, we use the algorithm in Section 3.1 and then finally output the minimum over all the clusters. The overall running time is *linear* for this step since each point belongs to at most 9 clusters.

Theorem 1. *Given a set P of n points in the plane, and a parameter k , one can compute, in expected linear time, a radius r , such that $r_{opt}(P, k) \leq r \leq 2r_{opt}(P, k)$.*

Once we have a 2-approximation r to $r_{opt}(P, k)$, using the algorithm of Theorem 1, we apply the exact algorithm of Matoušek [13] to each cluster of the grid $G_r(P)$ which contains more than k points.

Matoušek's algorithm has running time of $O(n \log n + nk)$ and space complexity $O(nk)$. By the choice of r , each cluster in which we apply the algorithm has $O(k)$ points. Thus the running time of the algorithm in each cluster is $O(k^2)$ and requires $O(k^2)$ space. The number of clusters which contain more than k points is $O(n/k)$. Hence the overall running time of our algorithm is $O(nk)$. Also the space requirement is $O(n + k^2)$.

Theorem 2. *Given a set P of n points in the plane, and a parameter k , one can compute, in expected $O(nk)$ time and space $O(n + k^2)$, the radius $r_{opt}(P, k)$, and a disk of this radius that covers k points of P .*

4 From Constant Approximation to $(1+\delta)$ -Approximation

Suppose r is a 2-approximation to $r_{opt}(P, k)$. Now if we construct $G_r(P)$ each grid cell contains less than $5k$ points of P (each grid cell can be covered fully by 5 circles of radius $r_{opt}(P, k)$). Furthermore, the smallest k -enclosing circle is covered by some grid cluster. We compute a $(1+\delta)$ -approximation to the radius of the minimal k enclosing circle in each grid cluster and output the smallest amongst them. The technique to compute $(1+\delta)$ -approximation when all the points belong to a particular grid cluster is as follows.

Let P_c be the set of points in a particular grid cluster with $k \leq |P_c| = O(k)$. Let R be a bounding square of the points of P_c . We partition R into a uniform grid G of size $\beta r \delta$, where β is an appropriately small constant. Next, snap every point of P_c into the closest grid point of G , and let P'_c denote the resulting point set. Clearly, $|P'_c| = O(1/\delta^2)$. Assume that we guess the radius $r_{opt}(P_c, k)$ up to a factor of $1+\delta$ (there are only $O(\log_{1+\delta} 2) = O(1/\delta)$ possible guesses), and let r' be the current guess. We need to compute for each point p of P'_c , how many points of P'_c are contained in $D(p, r')$. This can be done in $O((1/\delta) \log(1/\delta))$ time per point, by constructing a quadtree over the points of P'_c . Thus, computing a $\delta/4$ -approximation to the $r_{opt}(P'_c, k)$ takes $O((1/\delta^3) \log^2(1/\delta))$ time.

We repeat the above algorithm for all the clusters that have more than k points inside them. Clearly, the smallest disk computed is the required approximation. The running time is $O(n + n/(k\delta^3) \log^2(1/\delta))$. Putting this together with the algorithm of Theorem 1, we have:

Theorem 3. *Given a set P of n points in the plane, and parameters k and $\delta > 0$, one can compute, in expected*

$$O\left(n + n \cdot \min\left(\frac{1}{k\delta^3} \log^2 \frac{1}{\delta}, k\right)\right)$$

time, a radius r , such that $r_{\text{opt}}(P, k) \leq r \leq (1 + \delta)r_{\text{opt}}(P, k)$.

5 Conclusions

We presented a linear time algorithm that approximates up to a factor of two the smallest enclosing disk that contains at least k points, in the plane. This algorithm improves over previous results, and it can in some sense be interpreted as an extension of Rabin [15] closest pair algorithm to the clustering problem.

Getting similar results for other shape fitting problems, like the minimum radius cylinder in three dimensions, remains elusive. Current approaches for approximating it, in the presence of outliers, essentially reduces to the computation of the shortest vertical segment that stabs at least k hyperplanes. See [12] for the details. However, the results of Erickson and Seidel [9,8] imply that approximating the shortest vertical segment that stabs $d + 1$ hyperplanes takes $\Omega(n^d)$ time, under a reasonable computation model, thus implying that this approach is probably bound to fail if we are interested in a near linear time algorithm.

It would be interesting to figure out which of the shape fitting problems can be approximated in near linear time, in the presence of outliers, and which ones can't. We leave this as an open problem for further research.

Acknowledgments. The authors thank Alon Efrat and Edgar Ramos for helpful discussions on the problems studied in this paper.

References

1. P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. <http://www.uiuc.edu/~sariel/research/papers/01/fitting/>, 2002.
2. P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.
3. M. Bern and D. Eppstein. Approximation algorithms for geometric problems. In D. S. Hochbaum, editor, *Approximation algorithms for NP-Hard problems*, pages 296–345. PWS Publishing Company, 1997.
4. B. Chazelle. *The Discrepancy Method*. Cambridge University Press, 2000.

5. T. M. Chan. Low-dimensional linear programming with violations. In *Proc. 43th Annu. IEEE Sympos. Found. Comput. Sci.*, 2002. to appear.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press / McGraw-Hill, Cambridge, Mass., 2001.
7. D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Comput. Geom.*, 11:321–350, 1994.
8. J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM J. Comput.*, 28:1198–1214, 1999.
9. J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete Comput. Geom.*, 13:41–57, 1995.
10. A. Efrat, M. Sharir, and A. Ziv. Computing the smallest k -enclosing circle and related problems. *Comput. Geom. Theory Appl.*, 4:119–136, 1994.
11. S. Har-Peled and K. R. Varadarajan. Approximate shape fitting via linearization. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 66–73, 2001.
12. S. Har-Peled and Y. Wang. Shape fitting with outliers. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 29–38, 2003.
13. J. Matoušek. On enclosing k points by a circle. *Inform. Process. Lett.*, 53:217–221, 1995.
14. J. Matoušek. On geometric optimization with few violated constraints. *Discrete Comput. Geom.*, 14:365–384, 1995.
15. M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, New York, NY, 1976.
16. P. K. Agarwal, M. de Berg, J. Matoušek and O. Schwarzkopf. Constructing Levels in Arrangements and Higher Order Voronoi Diagrams *SICOMP*, 27:654–667, 1998.

The Minimum Generalized Vertex Cover Problem

Refael Hassin and Asaf Levin

Department of Statistics and Operations Research, Tel-Aviv University, Tel-Aviv
69978, Israel. {hassin,levinas}@post.tau.ac.il

Abstract. Let $G = (V, E)$ be an undirected graph, with three numbers $d_0(e) \geq d_1(e) \geq d_2(e) \geq 0$ for each edge $e \in E$. A solution is a subset $U \subseteq V$ and $d_i(e)$ represents the cost contributed to the solution by the edge e if exactly i of its endpoints are in the solution. The cost of including a vertex v in the solution is $c(v)$. A solution has cost that is equal to the sum of the vertex costs and the edge costs. The minimum generalized vertex cover problem is to compute a minimum cost set of vertices. We study the complexity of the problem when the costs $d_0(e) = 1$, $d_1(e) = \alpha$ and $d_2(e) = 0 \forall e \in E$ and $c(v) = \beta \forall v \in V$ for all possible values of α and β . We also provide a pair of 2-approximation algorithms for the general case.

1 Introduction

Given an undirected graph $G = (V, E)$ the MINIMUM VERTEX COVER PROBLEM is to find a minimum size vertex set $S \subseteq V$ such that for every $(i, j) \in E$ at least one of i and j belongs to S .

In the minimum vertex cover problem it makes no difference if we cover an edge by both its endpoints or by just one of its endpoints. In this paper we generalize the problem and an edge incurs a cost that depends on the number of its endpoints that belong to S .

Let $G = (V, E)$ be an undirected graph. For every edge $e \in E$ we are given three numbers $d_0(e) \geq d_1(e) \geq d_2(e) \geq 0$ and for every vertex $v \in V$ we are given a number $c(v) \geq 0$.

For a subset $S \subseteq V$ denote by $E(S) = E \cap (S \times S)$, $E(S, \bar{S}) = E \cap (S \times \bar{S})$, $c(S) = \sum_{v \in S} c(v)$, and for $i = 0, 1, 2$ $d_i(S) = \sum_{e \in E(S)} d_i(e)$ and $d_i(S, \bar{S}) = \sum_{e \in E(S, \bar{S})} d_i(e)$.

The MINIMUM GENERALIZED VERTEX COVER PROBLEM (GVC) is to find a vertex set $S \subseteq V$ that minimizes the cost $c(S) + d_2(S) + d_1(S, \bar{S}) + d_0(\bar{S})$. Thus, the value $d_i(e)$ represents the cost of the edge e if exactly i of its endpoints are included in the solution, and the cost of including a vertex v in the solution is $c(v)$.

Note that GVC generalizes the unweighted MINIMUM VERTEX COVER PROBLEM which is the special case with $d_0(e) = 1$, $d_1(e) = d_2(e) = 0 \forall e \in E$ and $c(v) = 1 \forall v \in V$.

An illustrative explanation for this problem is the following (see [6] and [3]): Let $G = (V, E)$ be an undirected graph. For each vertex $v \in V$ we can upgrade v at a cost $c(v)$. For each edge $e \in E$ $d_i(e)$ represents the cost of the edge e if exactly i of its endpoints are upgraded. The goal is to find a subset of upgraded vertices, such that the total upgrading and edge costs, is minimized.

Using this illustration, we will use the term *upgraded vertex* to denote a vertex that is included in the solution, and *non-upgraded vertex* to denote a vertex that is not included in the solution.

Paik and Sahni [6] presented a polynomial time algorithm for finding a minimum size set of upgraded vertices such that a given set of performance criteria will be met. Krumke, Marathe, Noltmeier, Ravi, Ravi, Sundaram and Wirth [3] considered the problem of a given budget that can be used to upgrade vertices and the goal is to upgrade a vertex set such that in the resulting network the minimum cost spanning tree is minimized.

When $d_0(e) = 1, d_1(e) = \alpha, d_2(e) = 0 \forall e \in E$ and $c(v) = \beta \forall v \in V$ we obtain the MINIMUM UNIFORM COST GENERALIZED VERTEX COVER PROBLEM (UGVC). Thus, the input to UGVC is an undirected graph $G = (V, E)$ and a pair of constants α (such that $0 \leq \alpha \leq 1$) and β . The cost of a solution $S \subseteq V$ for UGVC is $\beta|S| + |E(\bar{S})| + \alpha|E(S, \bar{S})|$.

The MAXIMIZATION VERSION OF GVC (Max-GVC) is defined as follows: given a graph $G = (V, E)$, three profit values $0 \leq p_0(i, j) \leq p_1(i, j) \leq p_2(i, j)$ for each edge $(i, j) \in E$, and an upgrade cost $c(v) \geq 0$ for each vertex $v \in V$. $p_k(i, j)$ denotes the profit from the edge (i, j) when exactly k of its endpoints are upgraded. The objective is to maximize the net profit, that is, the total profit minus the upgrading cost.

Our Results

- We study the complexity of UGVC for all possible values of α and β . The shaded areas in Figure 1 illustrate the polynomial-time solvable cases, whereas all the other cases are NP-hard. The analysis consists of eight lemmas. Lemmas 1-3 contain constructive proofs that the problem can be solved in polynomial time in the relevant regions, whereas Lemmas 4-8 contain reductions that prove the hardness of the problem in the respective regions. The numbers in each region refers to the lemma that provides a polynomial algorithm or proves the hardness of the problem in that region.
- We provide a 2-approximation $O(mn)$ -time algorithm for GVC based on linear programming relaxation.
- We provide another $O(m + n)$ -time 2-approximation algorithm.
- We show that Max-GVC is NP-hard and provide an $O(n^3)$ -time 2-approximation algorithm for Max-GVC.

2 The Complexity of UGVC

In this section we study the complexity of UGVC.

Lemma 1. *If $\frac{1}{2} \leq \alpha \leq 1$ then UGVC can be solved in polynomial time.*

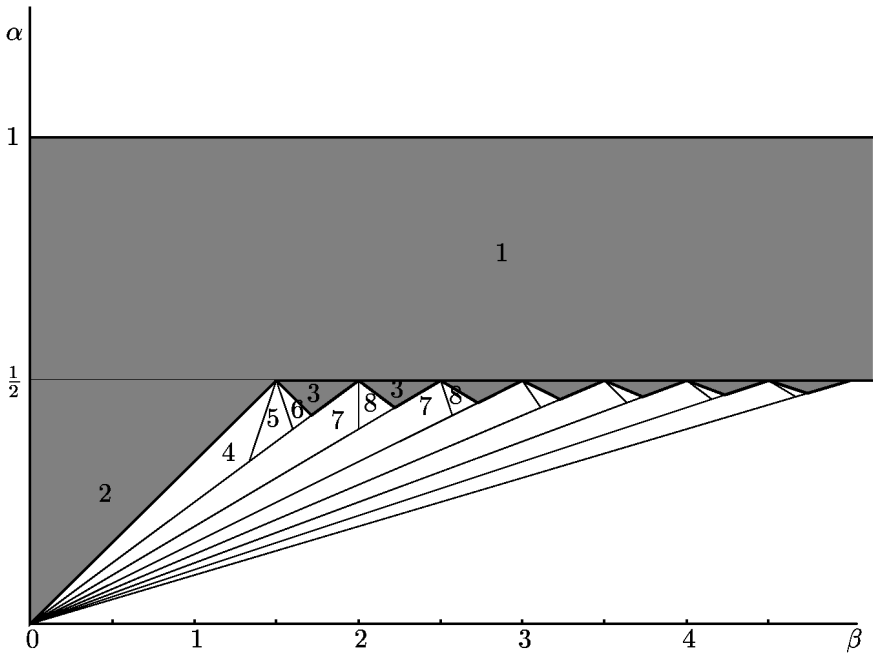


Fig. 1. The complexity of UGVC

Proof. The PROVISIONING PROBLEM was shown in [4] pages 125-127 to be solvable in polynomial time: Suppose there are n items to choose from, where item j costs $c_j \geq 0$. Also suppose there are m sets of items $S_1, S_2, \dots, S_m$. If all the items in set S_i are chosen, then a benefit of $b_i \geq 0$ is gained. The objective is to maximize the net benefit, i.e., total benefit gained minus total cost of items purchased.

If $\frac{1}{2} \leq \alpha \leq 1$ then UGVC is reducible to the provisioning problem as follows. The items are the vertices of the graph each has a cost of β . The sets are of two types: a single item $\{v\}$ for every vertex $v \in V$, and a pair $\{u, v\}$ of vertices for every edge $(u, v) \in E$. A set of a single vertex $\{v\}$ has a benefit of $(1 - \alpha)\deg(v)$ and a set that is a pair of vertices has a benefit of $2\alpha - 1 \geq 0$.

For a graph G , a *leaf* is a vertex with degree 1.

Lemma 2. *If $\alpha < \frac{1}{2}$ and $\beta \leq 3\alpha$ then UGVC can be solved in polynomial time.*

Proof. We first observe that upgrading one end of an edge saves $1 - \alpha$, and if also the other end of the edge is upgraded then the additional saving is α . By assumption $\alpha < \frac{1}{2}$ and therefore $\alpha < 1 - \alpha$.

By assumption $\beta \leq 3\alpha$, and therefore it is optimal to upgrade all the vertices whose degree is greater than or equal to 3. The analysis of the solution for the leaves and vertices with degree 2 is omitted.

Lemma 3. *If $\alpha < \frac{1}{2}$ and there exists an integer $d \geq 3$ such that $d(1 - \alpha) \leq \beta \leq (d + 1)\alpha$ then UGVC can be solved in polynomial time.*

Proof. Simply upgrade a vertex if and only if its degree is at least $d + 1$.

If Lemma 1, Lemma 2, and Lemma 3 can not be applied then UGVC is NP-hard. We will divide the proof into several cases.

Lemma 4. *If $\alpha < \frac{1}{2}$ and $3\alpha < \beta \leq 1 + \alpha$ then UGVC is NP-hard even when G is 3-regular.*

Proof. Assume that G is 3-regular and assume a solution to UGVC which upgrades k vertices. Because of the lemma's assumptions, if there is an edge $(u, v) \in E$ such that both u and v are not upgraded then it is better to upgrade u (resulting in an improvement of at least $1 - \alpha + 2\alpha - \beta = 1 + \alpha - \beta \geq 0$). Therefore, w.l.o.g. the solution is a vertex cover (if $\beta = 1 - \alpha$ then not all the optimal solutions are vertex covers, however, it is easy to transform a solution into a vertex cover without increasing the cost). Since there are $2|E| - 3k$ edges such that exactly one of their endpoints is upgraded, the cost of the solution is $\beta k + \alpha(2|E| - 3k) = k(\beta - 3\alpha) + 2\alpha|E|$. Since $\beta > 3\alpha$, the cost of the solution is a strictly monotone increasing function of k . Therefore, finding an optimal solution to UGVC for G is equivalent to finding a minimum vertex cover for G . The MINIMUM VERTEX COVER PROBLEM restricted to 3-regular graphs is NP-hard (see problems [GT1] and [GT20] in [1]).

Lemma 5. *If $\alpha < \frac{1}{2}$ and $1 + \alpha < \beta < 2 - \alpha$ then UGVC is NP-hard even when G is 3-regular.*

Proof. Assume that the input to UGVC with α, β satisfying the lemma's conditions, is a 3-regular graph $G = (V, E)$. By local optimality of the optimal solution for a vertex v , v is upgraded if and only if at least two of its neighbors are not upgraded: If v has at least two non-upgraded neighbors then upgrading v saves at least $2(1 - \alpha) + \alpha - \beta = 2 - \alpha - \beta > 0$; if v has at least two upgraded neighbors then upgrading v adds to the total cost at least $\beta - 2\alpha - (1 - \alpha) = \beta - (1 + \alpha) > 0$.

We will show that the following decision problem is NP-complete: given a 3-regular graph G and a number K , is there a solution to UGVC with cost at most K . The problem is clearly in NP. To show completeness we present a reduction from NOT-ALL-EQUAL-3SAT PROBLEM.

The NOT-ALL-EQUAL-3SAT is defined as follows (see [1]): given a set of clauses $S = \{C_1, C_2, \dots, C_p\}$ each with exactly 3 literals, is there a truth assignment such that each clause has at least one true literal and at least one false literal.

Given a set $S = \{C_1, C_2, \dots, C_p\}$ each with exactly 3 literals, construct a 3-regular graph $G = (V, E)$ as follows (see Figure 2, see the max-cut reduction in [7])

for similar ideas): For a variable x that appears in $p(x)$ clauses, G has $2p(x)$ vertices $A_1^x, \dots, A_{p(x)}^x, B_1^x, \dots, B_{p(x)}^x$ connected in a cycle $A_1^x, B_1^x, A_2^x, B_2^x, \dots, A_{p(x)}^x, B_{p(x)}^x, A_1^x$. In addition, for every clause C let G have six vertices $y_1^C, y_2^C, y_3^C, z_1^C, z_2^C, z_3^C$ connected in two triangles y_1^C, y_2^C, y_3^C and z_1^C, z_2^C, z_3^C . Each set of 3 vertices corresponds to the literals of the clause. If x occurs in a clause C , and let y_j^C and z_j^C correspond to x then we assign to this occurrence of x a distinct pair A_i^x, B_i^x (distinct i for each occurrence of x or $\bar{x}$) and we connect y_j^C to A_i^x and z_j^C to B_i^x . If $\bar{x}$ occurs in a clause C , and let y_j^C and z_j^C correspond to x then we assign to this occurrence of $\bar{x}$ a distinct pair A_i^x, B_i^x and we connect y_j^C to B_i^x and z_j^C to A_i^x .

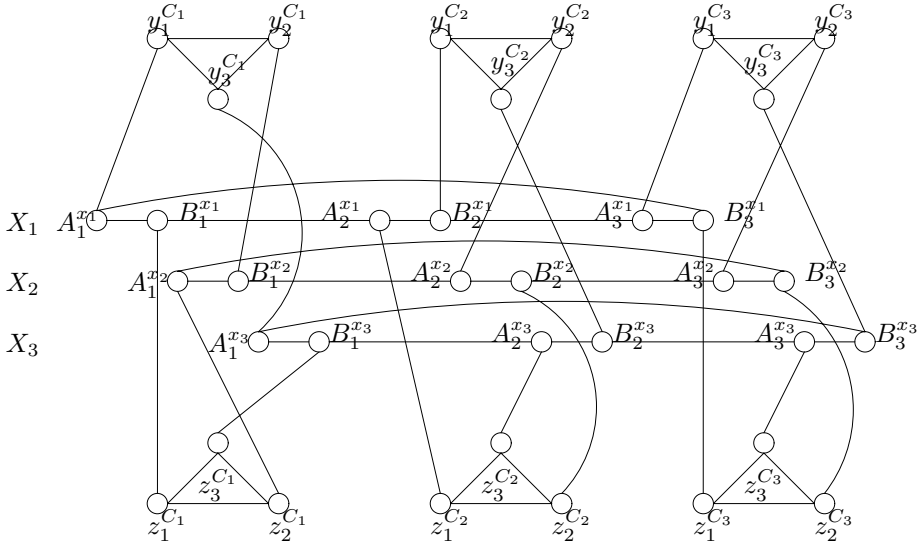


Fig. 2. The graph G obtained for the clauses $C_1 = x_1 \vee \bar{x}_2 \vee x_3$, $C_2 = \bar{x}_1 \vee x_2 \vee \bar{x}_3$, and $C_3 = x_1 \vee x_2 \vee \bar{x}_3$

Note that G is 3-regular.

For a 3-regular graph we charge the upgrading cost of an upgraded vertex to its incident edges. Therefore, the cost of an edge such that both its endpoints are upgraded is $\frac{2\beta}{3}$, the cost of an edge such that exactly one of its endpoints is upgraded is $\frac{\beta}{3} + \alpha$, and the cost of an edge such that none of its endpoints is upgraded is 1. Note that by the conditions on α and β , $\frac{\beta}{3} + \alpha < \frac{2\beta}{3}$ because by assumption $\beta \geq 1 + \alpha \geq 3\alpha$. Also, $\frac{\beta}{3} + \alpha < \frac{2-\alpha}{3} + \alpha = \frac{2}{3}(1 + \alpha) < 1$. Therefore, the cost of an edge is minimized if exactly one of its endpoints is upgraded.

We will show that there is an upgrading set with total cost of at most $(|E| - 2p)(\frac{\beta}{3} + \alpha) + p\frac{2\beta}{3} + p$ if and only if the NOT-ALL-EQUAL-3SAT instance can be satisfied.

Assume that S is satisfied by a truth assignment T . If $T(x) = TRUE$ then we upgrade B_i^x $i = 1, 2, \dots, p(x)$ and do not upgrade A_i^x $i = 1, 2, \dots, p(x)$. If $T(x) = FALSE$ then we upgrade A_i^x $i = 1, 2, \dots, p(x)$ and do not upgrade B_i^x $i = 1, 2, \dots, p(x)$. For a clause C we upgrade all the y_j^C vertices that correspond to TRUE literals and all the z_j^C vertices that correspond to FALSE literals. We note that the edges with either both endpoints upgraded or both not upgraded, are all triangle's edges. Note also that for every clause there is exactly one edge connecting a pair of upgraded vertices and one edge connecting a pair of non-upgraded vertices. Therefore, the total cost of the solution is exactly $(|E| - 2p)(\frac{\beta}{3} + \alpha) + p\frac{2\beta}{3} + p$.

Assume that there is an upgrading set U whose cost is at most $(|E| - 2p)(\frac{\beta}{3} + \alpha) + p\frac{2\beta}{3} + p$. Let $\bar{U} = V \setminus U$. Denote an upgraded vertex by U -vertex and a non-upgraded vertex by $\bar{U}$ -vertex. W.l.o.g. assume that U is a local optimum, and therefore a U -vertex has at most one U -neighbor and a $\bar{U}$ -vertex has at most one $\bar{U}$ -neighbor. Therefore, for a triangle y_1^C, y_2^C, y_3^C (z_1^C, z_2^C, z_3^C) at least one of its vertices is in U and at least one of its vertices is in $\bar{U}$. Therefore, in the triangle there is exactly one edge that connects either two U -vertices or two $\bar{U}$ -vertices and the two other edges connect a U -vertex to a $\bar{U}$ -vertex.

We will show that in G there are at least p edges that connect a pair of U -vertices and at least p edges that connect a pair of $\bar{U}$ -vertices. Otherwise there is a clause C such that for some j either y_j^C, z_j^C are both in U or both in $\bar{U}$. W.l.o.g. assume that y_j^C is connected to A_i^x and z_j^C is connected to B_i^x . Assume $y_j^C, z_j^C \in U$ ($y_j^C, z_j^C \in \bar{U}$) then by the local optimality of the solution, $A_i^x, B_i^x \in \bar{U}$ ($A_i^x, B_i^x \in U$), as otherwise y_j^C or z_j^C will have two U -($\bar{U}$)-neighbors and therefore we will not upgrade (will upgrade) them. Therefore, the edge (A_i^x, B_i^x) connects a pair of $\bar{U}$ (U) vertices. We charge every clause for the edges in the triangles corresponding to it that connect either two U -vertices or two $\bar{U}$ -vertices, and we also charge the clause for an edge (A_i^x, B_i^x) as in the above case. Therefore, we charge every clause for at least one edge that connects two U -vertices and for at least one edge that connects two $\bar{U}$ -vertices. These charged edges are all disjoint. Therefore, there are at least p edges that connect two U -vertices and at least p edges that connect two $\bar{U}$ -vertices.

Since the total cost is at most $(|E| - 2p)(\frac{\beta}{3} + \alpha) + p\frac{2\beta}{3} + p$, there are exactly p edges of each such type. Therefore, for every clause C for every j there is exactly one of the vertices y_j^C or z_j^C that is upgraded. Also note that for every variable x either $A_i^x \in U, B_i^x \in \bar{U} \forall i$ or $A_i^x \in \bar{U}, B_i^x \in U \forall i$. If $B_i^x \in U \forall i$ we assign to x the value TRUE and otherwise we assign x the value FALSE. We argue that this truth assignment satisfies S . In a clause C if $y_j^C \in U$ then its non-triangle neighbor is not upgraded and therefore, the literal corresponding to y_j^C is assigned a TRUE value. Similarly if $y_j^C \in \bar{U}$ the literal is assigned a FALSE value. Since in every triangle at least one vertex is upgraded and at least

one vertex is not upgraded there is at least one FALSE literal and at least one TRUE literal. Therefore, S is satisfied.

Lemma 6. *If $\alpha < \frac{1}{2}$, $2 - \alpha \leq \beta < 3(1 - \alpha)$ then UGVC is NP-hard even when G is 3-regular.*

Proof. Assume that G is 3-regular and assume a solution to UGVC which upgrades k vertices. Let $v \in V$, because of the lemma's assumptions if any of v 's neighbors is upgraded then not upgrading v saves at least $\beta - 2(1 - \alpha) - \alpha = \beta - (2 - \alpha) \geq 0$. Therefore, w.l.o.g. the solution is an independent set (if $\beta = 2 - \alpha$ then not all the optimal solutions are independent sets, however, it is easy to transform a solution into an independent set without increasing the cost). The cost of the solution is exactly $\beta k + 3k\alpha + (|E| - 3k) = |E| - k[3(1 - \alpha) - \beta]$. Since $3(1 - \alpha) > \beta$ the cost of the solution is strictly monotone decreasing function of k . Therefore, finding an optimal solution to UGVC for G is equivalent to finding an optimal independent set for G . The MAXIMUM INDEPENDENT SET PROBLEM restricted to 3-regular graphs is NP-hard (see problem [GT20] in [1]).

Lemma 7. *If $\alpha < \frac{1}{2}$ and $d\alpha < \beta \leq \min\{d\alpha + (d - 2)(1 - 2\alpha), (d + 1)\alpha\}$ for some integer $d \geq 4$ then UGVC is NP-hard.*

Proof. Let $G = (V, E)$ be a 3-regular graph that is an input to the MINIMUM VERTEX COVER PROBLEM. Since $d\alpha < \beta \leq d\alpha + (d - 2)(1 - 2\alpha)$, there is an integer k , $0 \leq k \leq d - 3$, such that $d\alpha + k(1 - 2\alpha) < \beta \leq d\alpha + (k + 1)(1 - 2\alpha)$.

We produce from G a graph $G' = (V'E')$ by adding k new neighbors (new vertices) to every vertex $v \in V$. From G' we produce a graph G'' by repeating the following for every vertex $v \in V$: add $d - k - 3$ copies of star centered at a new vertex with $d + 1$ leaves such that v is one of them and the other leaves are new vertices.

Since $\beta \leq (d + 1)\alpha$, w.l.o.g. in an optimal solution of UGVC on G'' every such center of a star is upgraded. Consider a vertex $u \in V'' \setminus V$ then u is either a center of a star or a leaf. If u is a leaf then since $\beta > \alpha$ then an optimal solution does not upgrade u .

In G'' every vertex from V has degree $3 + k + (d - k - 3) = d$ and in an optimal solution for the upgrading problem, at least one of the endpoints of every edge $(u, v) \in E$ is upgraded as otherwise u will have at least $k + 1$ non-upgraded neighbors, and since $\beta \leq d\alpha + (k + 1)(1 - 2\alpha)$, it is optimal to upgrade u .

Assume the optimal solution upgrades l vertices from V . The total cost of upgrading the l vertices and the cost of edges incident to vertices from V is $l\beta + lk\alpha + (n - l)k + (n - l)(d - k - 3)\alpha + (2|E| - 3l)\alpha = l[\beta + \alpha(k - d + k) - k] + n(k + (d - k - 3)\alpha) + 2|E|\alpha$. Since $\beta > k(1 - \alpha) + (d - k)\alpha$, the cost is strictly monotone increasing function of l . Therefore, to minimize the upgrading network cost is equivalent to finding a minimum vertex cover for G . Therefore, UGVC is NP-hard.

Lemma 8. *If $\alpha < \frac{1}{2}$ and $d\alpha + (d - 2)(1 - 2\alpha) \leq \beta < \min\{d\alpha + d(1 - 2\alpha), (d + 1)\alpha\}$ for some integer $d \geq 4$ then UGVC is NP-hard.*

Proof. Let $G = (V, E)$ be 3-regular graph that is an input to the MAXIMUM INDEPENDENT SET PROBLEM. Since $d\alpha + (d-2)(1-2\alpha) \leq \beta < d\alpha + d(1-2\alpha)$, $d\alpha + (d-k-1)(1-2\alpha) \leq \beta < d\alpha + (d-k)(1-2\alpha)$ holds for either $k = 0$ or for $k = 1$.

If $k = 1$ we add to every vertex $v \in V$ a star centered at a new vertex with $d+1$ leaves such that v is one of them. Since $\beta \leq (d+1)\alpha$, in an optimal solution the star's center is upgraded.

For every vertex in V we add $d-k-3$ new neighbors (new vertices). Consider a vertex $u \in V'' \setminus V$ then u is either a center of a star or a leaf. If u is a leaf then since $\beta \geq d\alpha + (d-2)(1-2\alpha) > 1-\alpha$, an optimal solution does not upgrade u .

Denote the resulting graph G' . The optimal upgrading set S in G' induces an independent set over G because if $u, v \in S \cap V$ and $(u, v) \in E$ then u has at least $k+1$ upgraded neighbors and therefore since $d\alpha + (d-k-1)(1-2\alpha) \leq \beta$, it is better not to upgrade u .

Assume the optimal solution upgrades l vertices from V . The total cost of upgrading the l vertices and the cost of edges incident to vertices from V is: $nk\alpha + (d-3-k)n + \frac{3n}{2} - l[k\alpha + (d-k)(1-\alpha) - \beta]$. Since $\beta < d\alpha + (d-k)(1-2\alpha)$, the cost is strictly monotone decreasing function of l , and therefore, it is minimized by upgrading a maximum independent set of G . Therefore, UGVC is NP-hard.

We summarize the results:

Theorem 1. *In the following cases UGVC is polynomial:*

1. If $\alpha \geq \frac{1}{2}$.
2. If $\alpha < \frac{1}{2}$ and $\beta \leq 3\alpha$.
3. If $\alpha < \frac{1}{2}$ and there exists an integer $d \geq 3$ such that $d(1-\alpha) \leq \beta \leq (d+1)\alpha$.

Otherwise, UGVC is NP-hard.

3 Approximation Algorithms

In this section we present two 2-approximation algorithms for the GVC problem. We present an approximation algorithm to GVC based on LP relaxation. We also present another algorithm with reduced time complexity for the special case where $d_0(e) - d_2(e) \geq 2(d_1(e) - d_2(e)) \forall e \in E$.

3.1 2-Approximation for GVC

For the following formulation we explicitly use the fact that every edge $e \in E$ is a subset $\{i, j\}$ where $i, j \in V$. Consider the following integer program (**GVCIP**):

$$\text{Min} \quad \sum_{i=1}^n c(i)x_i + \sum_{\{i,j\} \in E} \left(d_2(i,j)z_{ij} + d_1(i,j)(y_{ij} - z_{ij}) + d_0(i,j)(1 - y_{ij}) \right)$$

subject to :

$$y_{ij} \leq x_i + x_j \quad \forall \{i, j\} \in E$$

$$\begin{aligned}
y_{ij} &\leq 1 & \forall \{i, j\} \in E \\
z_{ij} &\leq x_i & \forall \{i, j\} \in E \\
x_i &\leq 1 & \forall i \in V \\
x_i, y_{ij}, z_{ij} &\in \{0, 1\} & \forall \{i, j\} \in E.
\end{aligned}$$

In this formulation: x_i is an indicator variable that is equal to 1 if we upgrade vertex i ; y_{ij} is an indicator variable that is equal to 1 if at least one of the vertices i and j is upgraded; z_{ij} is an indicator variable that is equal to 1 if both i and j are upgraded; $y_{ij} = 1$ is possible only if at least one of the variables x_i or x_j is equal to 1; $z_{ij} = 1$ is possible only if both x_i and x_j equal 1; If y_{ij} or z_{ij} can be equal to 1 then in an optimal solution they will be equal to 1 since $d_2(i, j) \leq d_1(i, j) \leq d_0(i, j)$. Denote by **GVCLP** the continuous (LP) relaxation of **GVCIP**.

Hochbaum [2] presented a set of Integer Programming problems denoted as **IP2** that contains **GVCIP**. For **IP2**, Hochbaum showed that the basic solutions to the LP relaxations of such problems are half-integral, and the relaxations can be solved using network flow algorithm in $O(mn)$ time. It is easy to get a direct proof of the first part for **GVCLP** and we omit the details.

The following is a 2-approximation algorithm:

1. Solve **GVCLP** using Hochbaum's [2] algorithm, and denote by x^*, y^*, z^* its optimal solution.
2. Upgrade vertex i if and only if $x_i^* \geq \frac{1}{2}$.

Theorem 2. *The above algorithm is an $O(mn)$ -time 2-approximation algorithm for GVC.*

Proof. Denote by $x_i^a = 1$ if we upgrade vertex i and $x_i^a = 0$ otherwise, $y_{ij}^a = \min\{x_i^a + x_j^a, 1\} = \max\{x_i^a, x_j^a\}$, and $z_{ij}^a = \min\{x_i^a, x_j^a\}$. The performance guarantee of the algorithm is derived by the following argument:

$$\begin{aligned}
& \sum_{i=1}^n c(i)x_i^a + \sum_{(i,j) \in E} (d_2(i, j)z_{ij}^a + d_1(i, j)(y_{ij}^a - z_{ij}^a) + d_0(i, j)(1 - y_{ij}^a)) \\
& \leq 2 \sum_{i=1}^n c(i)x_i^* + \sum_{(i,j) \in E} (d_2(i, j)z_{ij}^a + d_1(i, j)(y_{ij}^a - z_{ij}^a) + d_0(i, j)(1 - y_{ij}^a)) \\
& \leq 2 \sum_{i=1}^n c(i)x_i^* + \sum_{(i,j) \in E} (d_2(i, j)z_{ij}^* + d_1(i, j)(y_{ij}^* - z_{ij}^*) + d_0(i, j)(1 - y_{ij}^*)) \\
& < 2 \left(\sum_{i=1}^n c(i)x_i^* + \sum_{(i,j) \in E} (d_2(i, j)z_{ij}^* + d_1(i, j)(y_{ij}^* - z_{ij}^*) + d_0(i, j)(1 - y_{ij}^*)) \right)
\end{aligned}$$

The first inequality holds because we increase x_i by a factor which is at most 2. The second inequality holds because the second sum is a convex combination of $d_0(i, j)$, $d_1(i, j)$, and $d_2(i, j)$. Since $d_0(i, j) \geq d_1(i, j) \geq d_2(i, j)$,

$z_{ij}^a = \min\{x_i^a, x_j^a\} \geq \min\{x_i^*, x_j^*\} \geq z_{ij}^*$, and $1 - y_{ij}^a = \max\{1 - x_i^a - x_j^a, 0\} \leq \max\{1 - x_i^* - x_j^*, 0\} = 1 - y_{ij}^*$, the second inequality holds.

3.2 A Linear-Time 2-Approximation for GVC

Consider the following formulation **GVCLP'** obtained from **GVCLP** by exchanging variables: $X_i = x_i$, $Y_{ij} = 1 - y_{ij}$ and $Z_{ij} = 1 - z_{ij}$:

$$\begin{aligned} \text{Min} \quad & \sum_{i=1}^n c(i)X_i + \sum_{\{i,j\} \in E} \left(d_2(i,j) + [d_0(i,j) - d_1(i,j)]Y_{ij} + [d_1(i,j) - d_2(i,j)]Z_{ij} \right) \\ \text{subject to:} \quad & \\ X_i + X_j + Y_{ij} \geq 1 \quad & \forall \{i,j\} \in E \\ X_i + Z_{ij} \geq 1 \quad & \forall i \in V, \{i,j\} \in E \\ X_i, Y_{ij}, Z_{ij} \geq 0 \quad & \forall \{i,j\} \in E. \end{aligned}$$

The constraints $X_i, Y_{ij}, Z_{ij} \leq 1$ are clearly satisfied by an optimal solution, and we remove them from the formulation. The dual program of **GVCLP'** is the following (**DUALP**):

$$\begin{aligned} \text{Max} \quad & \sum_{\{i,j\} \in E} (\alpha_{ij} + \beta_{(i,j)} + \beta_{(j,i)}) \\ \text{subject to:} \quad & \\ \sum_{j: \{i,j\} \in E} (\alpha_{ij} + \beta_{(i,j)}) \leq c(i) \quad & \forall i \in V \quad (1) \\ \alpha_{ij} \leq d_0(i,j) - d_1(i,j) \quad & \forall \{i,j\} \in E \quad (2) \\ \beta_{(i,j)} + \beta_{(j,i)} \leq d_1(i,j) - d_2(i,j) \quad & \forall \{i,j\} \in E \quad (3) \\ \alpha_{ij}, \beta_{(i,j)}, \beta_{(j,i)} \geq 0 \quad & \forall \{i,j\} \in E. \end{aligned}$$

W.l.o.g. we assume that $d_2(i,j) = 0 \forall \{i,j\} \in E$ (otherwise, we can reduce $d_0(i,j)$, $d_1(i,j)$ and $d_2(i,j)$ by a common constant, and a 2-approximation for the transformed data will certainly be a 2-approximation for the original instance).

A feasible solution α, β for **DUALP** is a *maximal solution* if there is no other feasible solution α', β' for **DUALP** that differs from α, β and satisfies: $\alpha_{ij} \leq \alpha'_{ij}$, $\beta_{(i,j)} \leq \beta'_{(i,j)}$, $\beta_{(j,i)} \leq \beta'_{(j,i)}$ for every edge $\{i,j\} \in E$. A maximal solution for **DUALP** can be computed in linear time, by examining the variables in an arbitrary order, in each step we set the current variable to the largest size that is feasible (without changing any of the values of the variables that have already been set). The time complexity of this procedure is $O(m+n)$.

Theorem 3. *There is an $O(m+n)$ time 2-approximation algorithm for GVC.*

Proof. We show that the following is a 2-approximation algorithm.

1. Find a maximal solution for **DUALP**, and denote it by $\hat{\alpha}, \hat{\beta}$.

2. Upgrade vertex i if and only if its constraint in (1) is tight (i.e.,

$$\sum_{j:\{i,j\} \in E} (\hat{\alpha}_{ij} + \hat{\beta}_{(i,j)}) = c(i).$$

Denote by U the solution returned by the algorithm, and denote $\bar{U} = V \setminus U$. For each $\hat{\alpha}_{ij}$ and $\hat{\beta}_{(i,j)}$, we allocate a budget of twice its value. We show how the cost of U can be paid for using the total budget. Since $(\hat{\alpha}, \hat{\beta})$ is feasible to the dual problem **DUALP** and we assumed $d_2(i, j) = 0 \forall \{i, j\} \in E$, the cost of $(\hat{\alpha}, \hat{\beta})$ is a lower bound on the cost of a feasible solution to **GVCLP**, and therefore, the claim holds.

The following is the allocation of the total budget:

- $\hat{\alpha}_{u,v}$.
 - If $u, v \in \bar{U}$, then we allocate $\hat{\alpha}_{uv}$ to the edge (u, v) .
 - If $u \in U$ and $v \in \bar{U}$, then we allocate $\hat{\alpha}_{uv}$ to u .
 - If $u, v \in U$, then we allocate $\hat{\alpha}_{uv}$ to u and $\hat{\alpha}_{uv}$ to v .
- $\hat{\beta}_{(u,v)}$. We allocate $\hat{\beta}_{(u,v)}$ to u and $\hat{\beta}_{(u,v)}$ to (u, v) .

It remains to show that the cost of U was paid by the above procedure:

- $(u, v) \in E$ such that $u, v \in \bar{U}$. The edge (u, v) was paid $\hat{\alpha}_{uv} + \hat{\beta}_{(u,v)} + \hat{\beta}_{(v,u)}$. Since $u, v \in \bar{U}$, constraints (1) are not tight for u and v . Therefore, since $\hat{\alpha}, \hat{\beta}$ is a maximal solution, $\hat{\alpha}_{uv} = d_0(u, v) - d_1(u, v)$ and $\hat{\beta}_{(u,v)} + \hat{\beta}_{(v,u)} = d_1(u, v) - d_2(u, v)$. By assumption $d_2(u, v) = 0$, and therefore, the edge (u, v) was paid $d_0(u, v)$.
- $(u, v) \in E$ such that $u \in U, v \in \bar{U}$. Then, (u, v) was paid $\hat{\beta}_{(u,v)} + \hat{\beta}_{(v,u)}$. Note that since $v \in \bar{U}$ constraint (1) is not tight for v , and by the maximality of $\hat{\alpha}, \hat{\beta}$, we cannot increase $\hat{\beta}_{(v,u)}$. Therefore, constraint (3) is tight for $\{u, v\}$, and the edge was paid $d_1(u, v) - d_2(u, v) = d_1(u, v)$.
- $(u, v) \in E$ such that $u, v \in U$. We have to show that (u, v) was paid at least $d_2(u, v) = 0$, and this is trivial.
- $u \in U$. Then, u was paid $\hat{\alpha}_{uv} + \hat{\beta}_{(u,v)}$ by every edge $\{u, v\}$. Since $u \in U$, constraint (1) is tight for u . Therefore, $\sum_{v:\{u,v\} \in E} (\hat{\alpha}_{uv} + \hat{\beta}_{(u,v)}) = c(u)$. Therefore, u was paid $c(u)$.

3.3 Max-GVC

Consider the MAXIMIZATION VERSION OF GVC (Max-GVC).

Remark 1. Max-GVC is NP-hard.

Proof. This version is clearly NP-hard by the following straight forward reduction from the minimization version: for an edge $e \in E$ define $p_0(e) = 0$, $p_1(e) = d_0(e) - d_1(e)$, and $p_2(e) = d_0(e) - d_2(e)$. Then maximizing the net profit is equivalent to minimizing the total cost of the network.

If $p_2(i, j) - p_0(i, j) \geq 2[p_1(i, j) - p_0(i, j)]$ hold for every $(i, j) \in E$ then Max-GVC can be solved in polynomial time using the PROVISIONING PROBLEM (see Lemma 1): each vertex $v \in V$ is an item with cost $c(v) - \sum_{j:(i,j) \in E} [p_1(i, j) - p_0(i, j)]$, and each pair of vertices i, j is a set with benefit $p_2(i, j) - p_0(i, j) - 2[p_1(i, j) - p_0(i, j)] = p_2(i, j) - 2p_1(i, j) + p_0(i, j)$.

Theorem 4. *There is a 2-approximation algorithm for Max-GVC.*

Proof. Consider the following $O(n^3)$ -time algorithm:

- Solve the following PROVISIONING PROBLEM (see Lemma 1): each vertex $v \in V$ is an item with cost $c(v) - \sum_{j:(i,j) \in E} [p_1(i, j) - p_0(i, j)]$, and each pair of vertices i, j is a set with benefit $2p_2(i, j) - 2p_1(i, j) + p_0(i, j)$.

Consider the resulted solution S , and denote its value as a solution to the provisioning problem by $POPT$ and its net profit by APX . Denote the optimal value to the maximization of the net profit problem by OPT . Then the following inequalities hold: $APX \leq OPT \leq POPT$. For every upgraded vertex u we assign the increase of the net profit caused by upgrading u $c(u) - \sum_{v:(u,v) \in E} [p_1(u, v) - p_0(u, v)]$, and for a pair of adjacent upgraded vertices u, v we assigned net profit to the pair $\{u, v\}$ of $p_2(u, v) - 2p_1(u, v) + p_0(u, v)$. In this way we assigned all the net profit beside $\sum_{(i,j) \in E} p_0(i, j)$ which is a positive constant.

Since each set of items incurs a benefit of at most twice its assigned net profit. then $2APX \geq POPT$. Therefore, the algorithm is a 2-approximation algorithm.

References

1. M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, 1979.
2. D. S. Hochbaum, "Solving integer programs over monotone inequalities in three variables: A framework for half integrality and good approximations," *European Journal of Operational Research*, **140**, 291–321, 2002.
3. S. O. Krumke, M. V. Marathe, H. Noltemeier, R. Ravi, S. S. Ravi, R. Sundaram, and H. C. Wirth, "Improving minimum cost spanning trees by upgrading nodes", *Journal of Algorithms*, **33**, 92–111, 1999.
4. E. L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, 1976.
5. G. L. Nemhauser and L. E. Trotter, Jr., "Vertex packing: structural properties and algorithms", *Mathematical Programming*, **8**, 232–248, 1975.
6. D. Paik, and S. Sahni, "Network upgrading problems", *Networks*, **26**, 45–58, 1995.
7. M. Yannakakis, "Edge deletion problems", *SIAM J. Computing*, **10**, 297–309, 1981.

An Approximation Algorithm for MAX-2-SAT with Cardinality Constraint

Thomas Hofmeister

Informatik 2, Universität Dortmund, 44221 Dortmund, Germany
th01@ls2.cs.uni-dortmund.de

Abstract. We present a randomized polynomial-time approximation algorithm for the MAX-2-SAT problem in the presence of an extra cardinality constraint which has an asymptotic worst-case ratio of 0.75. This improves upon the previously best approximation ratio 0.6603 which was achieved by Bläser and Manthey [BM]. Our approach is to use a solution obtained from a linear program which we first modify greedily and to which we then apply randomized rounding. The greedy phase guarantees that the errors introduced by the randomized rounding are not too large, an approach that might be interesting for other applications as well.

1 Introduction and Preliminaries

In the MAXSAT problem, we are given a set of clauses. The problem is to find an assignment $a \in \{0, 1\}^n$ to the variables $x_1, \dots, x_n$ which satisfies as many of the clauses as possible. The MAX- k -SAT problem is the special case of MAXSAT where all input clauses have length at most k . It is already NP-hard for $k=2$, hence one has to be satisfied with approximation algorithms. An approximation algorithm for a satisfiability problem is said to have worst-case (approximation) ratio α if on all input instances, it computes an assignment which satisfies at least $\alpha \cdot OPT$ clauses when OPT is the maximum number of clauses simultaneously satisfiable.

Approximation algorithms for MAXSAT are well-studied. On the positive side, a polynomial-time approximation algorithm is known which is based on the method of semidefinite programming and which achieves a worst-case approximation ratio of 0.7846. For this result and an overview of the previously achieved ratios, we refer the reader to the paper by Asano and Williamson [AW]. They also present an algorithm with an approximation ratio that is *conjectured* to be 0.8331.

Simpler algorithms which are based on linear programming (“LP”) combined with randomized rounding achieve a worst-case ratio of 0.75, see the original paper by Goemans and Williamson [GW1] or the books by Motwani/Raghavan ([MR], Chapter 5.2) or Vazirani ([V], Chapter 16).

On the negative side, we mention only that Håstad [H] (Theorem 6.16) showed that a polynomial-time approximation algorithm for MAX-2-SAT with worst-case approximation ratio larger (by a constant) than $21/22 \approx 0.955$ would imply $P=NP$.

Sometimes, it is desirable to reduce the space of feasible assignments $x \in \{0, 1\}^n$ by extra constraints. The reason is that more problems can be transformed into such finer-grained satisfiability problems. The constraints which we consider in this paper are *cardinality* constraints, i.e., constraints that can be written as $x_1 + \dots + x_n = T$, where T is an integer. We remark that while we consider the cardinality constraint to be an equality, other papers prefer to have an inequality “ $\leq T$ ” instead. It should be clear that the result obtained in our paper also extends to this alternative definition as the algorithm only needs to be applied for $T' = 0, \dots, T$ if necessary.

Recently, cardinality-constrained variants of known NP-hard problems have obtained some attention, see e.g. [S,AS,FL,BM]. While Sviridenko in [S] considers the problem “MAXSATCC” which is the constrained variant of MAXSAT, Ageev and Sviridenko [AS] investigate the constrained variants of the MAX-CUT and MAXCOVER problems. Feige and Langberg have shown in [FL] that a semidefinite programming approach can improve the approximation ratio for some cardinality-constrained graph problems (among them the variants of MAX-CUT and VERTEX COVER). The MAX-2-SATCC problem which we consider in this paper was also considered before, in the paper by Bläser and Manthey [BM]. Before we describe some of the results, we start with some definitions.

Definition 1. *Given n Boolean variables $x_1, \dots, x_n$, an assignment to those variables is a vector $a = (a_1, \dots, a_n) \in \{0, 1\}^n$.*

A literal is either a variable x_i or its negation $\bar{x}_i$. In the first case, the literal is called positive, in the second, it is called negative.

A clause C of length k is a disjunction $C = l_1 \vee l_2 \vee \dots \vee l_k$ of literals.

A clause is called positive, if it only contains positive literals, negative, if it only contains negative literals, and pure if it is positive or negative.

A clause that is not pure will also be called mixed.

We assume in the following (without loss of generality) that each clause we are dealing with contains no variable twice, since it could be shortened otherwise. For a fixed constant k , the problems MAX- k -SAT and MAX- k -SATCC (“CC” being shorthand for “cardinality constraint”) are defined as follows:

Input: A set $\{C_1, \dots, C_m\}$ of clauses each of which has length at most k . For the MAX- k -SATCC problem, an integer T is also part of the input.

Problem MAX- k -SAT: Let $A = \{0, 1\}^n$. Find an assignment $a \in A$ which satisfies as many of the clauses as possible.

Problem MAX- k -SATCC: Let $A = \{a \in \{0, 1\}^n \mid \#a = T\}$, where $\#a$ denotes the number of ones in a . Find an assignment $a \in A$ which satisfies as many of the clauses as possible.

We note that we are considering the “unweighted” case of the problems, i.e., the input to the problems is a *set* of clauses and not a *list* of clauses.

It is well-known that already the MAX-2-SAT problem is NP-hard and since MAX-2-SAT can be solved by at most $n + 1$ invocations of MAX-2-SATCC, this

problem is NP-hard as well. Due to the negative result by Håstad mentioned above, it is also difficult to approximate beyond a ratio of $21/22$.

The algorithm which we describe is based on an approximation algorithm for MAXSAT by Goemans and Williamson [GW1] which uses linear programming and randomized rounding to achieve an approximation ratio 0.75. We will later on refer to this algorithm as the “LP-based approximation algorithm”. Its worst-case ratio is the same when we restrict the input to MAX-2-SAT instances.

Looking at this approximation algorithm, one might get the impression that the extra cardinality constraint does not make the problem much harder since it is easy to integrate the constraint into a linear program. Nevertheless, there is a clear hint that cardinality constraints can render satisfiability problems somewhat harder. For example, a polynomial-time algorithm for MAXSATCC with an approximation ratio larger (by a constant) than $1 - (1/e) \approx 0.632$ would mean that $\text{NP} \subseteq \text{DTIME}(n^{O(\log \log n)})$, see the paper by Feige [F], as we could approximate the SETCOVER problem to a ratio $c \cdot \ln n$ with $c < 1$. This is in well-marked contrast to the fact that there are polynomial-time approximation algorithms for MAXSAT with worst-case ratio larger than 0.78.

An algorithm achieving the above-mentioned best possible ratio $1 - (1/e)$ for MAXSATCC was given in [S] where the natural question is posed whether for MAX- k -SATCC, k fixed, better approximation ratios can be achieved.

A first answer to this question was given in [BM], where for the MAX-2-SATCC problem a polynomial-time approximation algorithm with worst-case ratio 0.6603 is described. We improve upon this result by designing a randomized polynomial-time algorithm which on input clauses $C_1, \dots, C_m$ and input number T computes an assignment z' which has exactly T ones. The number G of clauses that z' satisfies has the property that

$$E[G] \geq 3/4 \cdot \text{OPT}_{CC} - o(\text{OPT}_{CC}),$$

where $E[\cdot]$ denotes the expected value of a random variable and where OPT_{CC} is the maximum number of clauses which can simultaneously be satisfied by an assignment with exactly T ones.

With respect to the usual definitions, this means that our randomized approximation algorithm has an asymptotic worst-case ratio of $3/4$.

Our approach works as follows: As in the LP-based algorithm for MAXSAT, we first transform the given MAX-2-SAT instance into a linear program which can be solved in polynomial time, we only add the extra cardinality constraint to the linear program. The solution of the linear program yields n parameters $y_1^*, \dots, y_n^*$ with $0 \leq y_i^* \leq 1$ for all $i = 1, \dots, n$.

The LP-based algorithm for the general MAXSAT problem proceeds by applying randomized rounding to the y_i^* . On MAX-2-SAT instances, it can be shown that the so produced $\{0, 1\}$ -solutions on the average satisfy at least $(3/4) \cdot \text{OPT}$ of the clauses, where OPT is the value of the optimal MAX-2-SAT solution.

For MAX-2-SATCC, directly applying randomized rounding is prohibitive since the number of ones in the so obtained vector could be too far off the desired number T of ones and correcting the number of ones by flipping some bits in the vector might change the number of satisfied clauses too much.

Thus, our approach is to apply a technique that is called “pipage rounding” in [AS] as a preprocessing step and to *then* apply the normal randomized rounding to some remaining variables. We will see that the extra preprocessing step leaves us with a problem where we are better able to control the error term which is introduced by randomized rounding. The approach we use might be interesting in its own right since it shows that randomized rounding, which is an approach used in several contexts, can be improved by a greedy preprocessing phase.

2 Linear Programming and Randomized Rounding

We start by describing the standard approach of transforming a MAXSAT instance into a linear program which is used in the LP-based approximation algorithm.

A clause $C = l_1 \vee \dots \vee l_k$ is arithmetized by replacing negative literals $\bar{x}_i$ by $(1 - x_i)$ and replacing “ $\vee$ ” by “+”. E.g., $x_1 \vee \bar{x}_2$ is transformed into $x_1 + (1 - x_2)$. Thus, each clause C is transformed into a linear expression $\text{lin}(C)$. The linear program obtained from a set of clauses $\{C_1, \dots, C_m\}$ is as follows:

$$\begin{aligned} &\textbf{maximize} && \sum_{j=1}^m z_j \\ &\textbf{subject to} && \text{lin}(C_j) \geq z_j \quad \text{for all } j = 1, \dots, m. \\ &&& 0 \leq y_i, z_j \leq 1 \quad \text{for all } i = 1, \dots, n, \quad j = 1, \dots, m. \end{aligned}$$

Assume that $z_1^*, \dots, z_m^*, y_1^*, \dots, y_n^*$ is the optimal solution of this linear program and that the value of the objective function on this solution is OPT_{LP} . Then $\text{OPT}_{LP} \geq \text{OPT}$, where OPT is the maximum number of clauses simultaneously satisfiable by an assignment. The parameters $y_1^*, \dots, y_n^*$ are used for randomized rounding:

Randomized rounding with parameters $p_1, \dots, p_n$ randomly selects an assignment $a = (a_1, \dots, a_n) \in \{0, 1\}^n$ by choosing $a_i = 1$ with probability p_i and $a_i = 0$ with probability $1 - p_i$, independently for all $i = 1, \dots, n$.

For each clause C , there is a certain probability $P_C(p_1, \dots, p_n)$ that the clause is satisfied by randomized rounding with parameters $p_1, \dots, p_n$. It is easy to see that for every clause C of length k , P_C is a (multivariate) polynomial of degree k . E.g.:

$$\begin{aligned} C = x_1 \vee \bar{x}_2 &\Rightarrow P_C(p_1, \dots, p_n) = 1 - (1 - p_1) \cdot p_2 = 1 - p_2 + p_1 p_2. \\ C = \bar{x}_1 \vee \bar{x}_2 &\Rightarrow P_C(p_1, \dots, p_n) = 1 - p_1 p_2. \end{aligned}$$

Note that for 0-1-valued parameters, i.e., in the case that $p_1, \dots, p_n$ is an assignment, P_C yields the value 1 if the clause C is satisfied and 0 otherwise. For our purposes, it is also important to note the following: If C is a *pure* clause of length 2, then $P_C(p_1, p_2, \dots, p_n)$ is a polynomial in which the highest degree

monomial has a negative coefficient -1 while for a *mixed* clause, the corresponding coefficient is $+1$.

For a MAXSAT instance consisting of m clauses $C_1, \dots, C_m$, the following function F describes the expected number of satisfied clauses if an assignment is chosen according to randomized rounding with $p_1, \dots, p_n$.

$$F(p_1, \dots, p_n) := \sum_{i=1}^m P_{C_i}(p_1, \dots, p_n).$$

If all clauses C_j are of length at most 2, the analysis of the LP-based MAXSAT algorithm shows that $P_{C_j}(y_1^*, \dots, y_n^*) \geq 3/4 \cdot z_j^*$, hence

$$F(y_1^*, \dots, y_n^*) \geq (3/4) \cdot \sum_{j=1}^m z_j^* = (3/4) \cdot OPT_{LP} \geq (3/4) \cdot OPT.$$

A cardinality constraint $\sum_{i=1}^n x_i = T$ is a linear constraint and can easily be added to the linear program. We obtain a solution $y_1^*, \dots, y_n^*, z_1^*, \dots, z_m^*$ in polynomial time. Again, it holds that $F(y_1^*, \dots, y_n^*) \geq (3/4) \cdot \sum_{j=1}^m z_j^* \geq (3/4) \cdot OPT_{CC}$, where OPT_{CC} is the maximum number of clauses which can simultaneously be satisfied by an assignment with exactly T ones.

We will use the function F to guide us in the search for a good assignment. The solution of the linear program gives a good enough “starting point”.

Randomized rounding apparently cannot be applied directly since it can yield vectors with a number of ones that is “far away” from the desired number T . Repairing this by flipping some of the bits might change the F -value too much. Our algorithm starts with the solution $y^* = (y_1^*, \dots, y_n^*)$ of the linear program (with the extra constraint) and applies a greedy preprocessing phase to the parameters. We obtain a new vector (which we still call y^*) and consider those positions in y^* in more detail that are not yet 0-1-valued. Call this set of positions U : Due to the preprocessing phase, we have extra information on the mixed clauses that exist on the variables corresponding to the positions in U . We then show that randomized rounding performed with those variables introduces an error term which is not too large.

3 Randomized Rounding with Preprocessing

Our algorithm works as follows. We first transform the given set of clauses together with the cardinality constraint into a linear program, as described in the previous section. By solving the linear program, we obtain a vector $y^* = (y_1^*, \dots, y_n^*) \in [0, 1]^n$ which has the property that $F(y^*) \geq (3/4) \cdot OPT_{CC}$ and $\sum_{i=1}^n y_i^* = T$.

We use the vector y^* and modify y^* in three successive phases. First, we apply a greedy preprocessing phase where we consider pairwise positions in y^* that are both non-integer. A similar pairwise modification has already been used in [AS] where it is named a “pipage step”. Namely, in order to keep the sum of

all y_i^* unchanged, we can change two positions by increasing one of them and decreasing the other by the same amount. This can be done until one of them assumes either the value 0 or 1. The first phase applies such changes if they increase (or leave unchanged) the value F .

The second phase starts if no such changes can be applied anymore. It applies randomized rounding to the remaining non-integer positions. Since this randomized rounding can produce an assignment with a number of ones which is different from T , we need a third, “correcting” phase.

In the description of the algorithm, we need the set of positions in y^* that are non-integer, i.e., $U(y^*) := \{i \in \{1, \dots, n\} \mid y_i^* \notin \{0, 1\}\}$.

Phase 1: Greedy Preprocessing

The following two rules are applicable to pairs of positions in $U(y^*)$. Apply the rules in any order until none of them is applicable.

Rule 1a: If there is a pair $i \neq j$ with $i, j \in U(y^*)$ and $S := y_i^* + y_j^* \leq 1$, check whether changing (y_i^*, y_j^*) to $(0, S)$ or to $(S, 0)$ increases (or leaves unchanged) the F -value. If so, apply the change to y^* .

Rule 1b: Similar to rule 1a, but for the case that $S := y_i^* + y_j^* > 1$. I.e., we have to check $(1, S - 1)$ and $(S - 1, 1)$.

Phase 2: Randomized rounding

Phase 1 yields a vector $y^* = (y_1^*, \dots, y_n^*) \in [0, 1]^n$.

If $U(y^*)$ is empty, then the algorithm can stop with output $result := y^*$.

Otherwise, we may assume for notational convenience that $U(y^*) = \{1, \dots, a\}$ and that $y_{a+1}^*, \dots, y_n^*$ are already 0–1-valued. Define $s := \sum_{i=1}^a y_i^*$. Since $s = T - \sum_{i=a+1}^n y_i^*$, we know that s is an integer.

Construct a vector $z \in \{0, 1\}^a$ as follows: For $i = 1, \dots, a$, set $z_i := 1$ with probability y_i^* and $z_i := 0$ with probability $1 - y_i^*$, for all i independently.

Phase 3: Correcting

If the number of ones in z is what it should be, i.e., $\#z = s$, then this phase stops with $z' := z$. Otherwise, we correct z as follows:

If $\#z > s$, then we arbitrarily pick $\#z - s$ positions in z which we switch from one to zero to obtain a vector z' with s ones.

If $\#z < s$, then we arbitrarily pick $s - \#z$ positions in z which we switch from zero to one to obtain a vector z' with s ones.

Finally, the algorithm outputs the assignment $result := (z'_1, \dots, z'_a, y_{a+1}^*, \dots, y_n^*)$.

The number of ones in $result$ is T . This is true because $\sum_{i=1}^n y_i^* = T$ before phase 1. This sum is not changed by the application of the rules in phase 1. Finally, after phase 1 and also after phases 2 and 3, the sum $\sum_{i=1}^a y_i^*$ is s , hence $result$ contains $s + (T - s) = T$ ones.

The running time of the algorithm is of course polynomial, since the application of a rule in phase 1 decreases $|U(y^*)|$, so the rules are applicable at most n times. The running time is dominated by the time needed for solving the linear program.

Analyzing the Algorithm

By the way the rules work, after phase 1, we still have a vector y^* with $\sum_{i=1}^n y_i^* = T$ and $F(y_1^*, \dots, y_n^*) \geq (3/4) \cdot OPT_{CC}$.

Note that since we are dealing with the MAX-2-SATCC problem, the monomials in F have length at most 2. Since phases 2 and 3 leave positions $a+1$ to n , i.e. $y_{a+1}^*, \dots, y_n^*$, unchanged (which are 0-1-valued), we can fix the corresponding parameters in our objective function F and consider it as being dependent on the first a positions only, i.e., we can write (for some integer constants $d_{i,j}$, c_i and d):

$$\begin{aligned} F_a(x_1, \dots, x_a) &:= F(x_1, \dots, x_a, y_{a+1}^*, \dots, y_n^*) \\ &= \sum_{1 \leq i < j \leq a} d_{i,j} \cdot x_i \cdot x_j + \sum_{i=1}^a c_i \cdot x_i + d. \end{aligned}$$

For notational convenience (in order to avoid case distinctions), we define for arbitrary $k \neq l$ that $d_{\{k,l\}} := d_{\min\{k,l\}, \max\{k,l\}}$.

Using simple calculus, we are now able to show that after phase 1, certain bounds on the coefficients in the remaining objective function hold:

Lemma 1. *If $F_a(x_1, \dots, x_a) = \sum_{1 \leq i < j \leq a} d_{i,j} \cdot x_i \cdot x_j + \sum_{i=1}^a c_i \cdot x_i + d$ is the objective function that we are left with after phase 1, then the following holds:*

- $d \geq 0$.
- $1 \leq d_{i,j} \leq 2$ for all $1 \leq i < j \leq a$.
- $c_i - c_j \leq a$ for all $i, j \in \{1, \dots, a\}$.

Proof. F counts an expected number of satisfied clauses, which can not be negative. Hence $F_a(0, \dots, 0) = d \geq 0$. For the proof of the other two properties, we will exploit the following well-known property of a real-valued function f which is defined on an interval $[l, r]$ (let f' and f'' denote the first and second derivatives): When one of the properties a)–c) is fulfilled for all $x \in [l, r]$, then f assumes its maximum on the interval $[l, r]$ at one of the endpoints of the interval: a) $f'(x) \leq 0$ b) $f'(x) \geq 0$ c) $f''(x) > 0$.

We know that $d_{i,j} \leq 2$, since every term $x_i x_j$ in F (and thus F_a) is generated by a clause of length two on the variables x_i and x_j . Only *mixed* clauses on x_i and x_j have a positive coefficient, namely $+1$, on $x_i x_j$. The input clauses contain at most two mixed clauses on x_i and x_j , since by definition, duplicate clauses in the input are not allowed. Hence, $d_{i,j} \leq 2$.

We now show that $d_{i,j} > 0$ by proving that otherwise, one of the rules would be applicable. Since $d_{i,j}$ is an integer, it follows that $d_{i,j} \geq 1$.

Consider a pair $i \neq j$ of positions. The rules in phase 1 can change them while maintaining their sum S . In order to investigate the effect of these rules, we define the function $H(x)$ as follows:

$$H(x) := F_a(y_1^*, y_2^*, \dots, \underbrace{x}_{\text{position } i}, \dots, \underbrace{S-x}_{\text{position } j}, \dots, y_a^*),$$

i.e., we fix all positions except for positions i and j and set the i -th position to x and the j -th position in such a way that their original sum $S = y_i^* + y_j^*$ is maintained. The first and second derivatives of H with respect to x are:

$$H'(x) = \sum_{k \in \{1, \dots, a\} \setminus \{i, j\}} (d_{\{i, k\}} - d_{\{j, k\}}) \cdot y_k^* + d_{\{i, j\}} \cdot S - 2 \cdot d_{\{i, j\}} \cdot x + (c_i - c_j).$$

$$H''(x) = -2 \cdot d_{\{i, j\}}.$$

When $d_{\{i, j\}} = 0$, then the first derivative does not depend on x , hence is a constant and either a) or b) holds. When $d_{\{i, j\}} < 0$, then the second derivative is larger than zero and c) is fulfilled. In both cases, H assumes its maximum at one of the endpoints. Since positions i and j are from $U(y^*)$, this would mean that either rule 1a or rule 1b would be applicable. But after phase 1, no such rule is applicable, hence $d_{i, j} > 0$ for all $i < j$.

In order to bound $c_i - c_j$, we observe that since $1 \leq d_{\{k, l\}} \leq 2$ for all $k \neq l$, we have $\sum_{k \in \{1, \dots, a\} \setminus \{i, j\}} (d_{\{i, k\}} - d_{\{j, k\}}) \cdot y_k^* \geq -(a-2)$ as well as

$$\begin{aligned} d_{\{i, j\}} \cdot S - 2 \cdot d_{\{i, j\}} \cdot x &= d_{\{i, j\}} \cdot (S - 2x) \\ &\geq d_{\{i, j\}} \cdot (-x) \geq -2. \end{aligned}$$

This shows that $H'(x) \geq -(a-2) - 2 + (c_i - c_j) = -a + (c_i - c_j)$.

If $c_i - c_j > a$, then the first derivative is larger than zero, i.e., by arguments analogous to the ones above, either rule 1a or rule 1b would be applicable. $\square$

(We remark that $c_i - c_j$ could also be bounded in terms of s , but for our purposes, the above bound is enough, as we will see below.)

By renumbering the variables if necessary, we can assume w.l.o.g. that $c_1 \geq c_2 \geq \dots \geq c_a$ holds. We can rewrite the objective function as follows:

$$F_a(x_1, \dots, x_a) = \sum_{1 \leq i < j \leq a} d_{i, j} \cdot x_i \cdot x_j + \sum_{i=1}^a (c_i - c_a) \cdot x_i + \left(\sum_{i=1}^a x_i \right) \cdot c_a + d.$$

Let G be the following function (which just omits some of the terms in F_a):

$$G(x_1, \dots, x_a) := \sum_{1 \leq i < j \leq a} d_{i, j} \cdot x_i \cdot x_j + \sum_{i=1}^a (c_i - c_a) \cdot x_i.$$

By Lemma 1, and by the renumbering of the variables, we know that $1 \leq d_{i, j} \leq 2$ as well as $0 \leq (c_i - c_a) \leq a$. This means that G is a monotone function.

We now analyze the effect of phases 2 and 3. Define the following sets of vectors A_{int} and A_{real} with $A_{int} \subseteq A_{real}$:

$$A_{int} := \{w \in \{0, 1\}^a \mid \sum_{i=1}^a w_i = s\} \quad \text{and}$$

$$A_{real} := \{w \in [0, 1]^a \mid \sum_{i=1}^a w_i = s\}.$$

At the beginning of phase 2, we have a vector y^* to which we apply changes in the first a positions. Let $v^* := (y_1^*, \dots, y_a^*) \in A_{real}$.

From this v^* , we construct a vector $z' \in A_{int}$. The following lemma shows that, on the average, $F_a(v^*)$ and $F_a(z')$ are not too far apart.

Lemma 2. *Let $v^* \in A_{real}$ be given. Phases 2 and 3, when started with v^* , yield a vector $z' \in A_{int}$ with the property that*

$$E[F_a(z')] \geq F_a(v^*) - o(a^2).$$

Proof. It is clearly enough to show that

$$E[G(z')] \geq G(v^*) - o(a^2)$$

since for any vectors $w \in A_{real}$, $F_a(w)$ and $G(w)$ differ exactly by the same constant (namely $d + s \cdot c_a$).

The randomized rounding phase first computes a vector z . By the way the rounding is performed, we have

$$E[\#z] = s \quad \text{as well as} \quad E[G(z)] = G(v^*).$$

If $\#z \leq s$, then it follows that $G(z') \geq G(z)$. This is because G is monotone and because z' is obtained from z by switching zeroes to ones.

If $\#z \geq s$, then it holds that

$$G(z') \geq G(z) - (\#z - s) \cdot 3a.$$

The reason for this is that $d_{i,j} \leq 2$ as well as $(c_i - c_a) \leq a$, hence changing a 1 to a 0 can change the G -value by at most $3a$. Thus, in both cases, we can write

$$G(z') \geq G(z) - |\#z - s| \cdot 3a.$$

By linearity of expectation, we can estimate:

$$E[G(z')] \geq E[G(z)] - E[|\#z - s|] \cdot 3a = G(v^*) - E[|\#z - s|] \cdot 3a.$$

In order to estimate $E[|\#z - s|]$, we apply Jensen's inequality (which states that " $E[Y] \leq \sqrt{E[Y^2]}$ ") to $Y := |\#z - s|$ and use $V[X] = E[(X - E[X])^2]$ to denote the variance of the random variable X . We obtain:

$$E[|\#z - s|] \leq \sqrt{E[(\#z - s)^2]} = \sqrt{E[(\#z - E[\#z])^2]} = \sqrt{V[\#z]}.$$

$\#z$ is the sum of independent Bernoulli-variables $z_1, \dots, z_a$, hence $V[\#z] = V[z_1] + \dots + V[z_a]$ as well as $V[z_i] = \text{Prob}(z_i = 1) \cdot (1 - \text{Prob}(z_i = 1)) \leq 1/4$, i.e., $V[\#z] \leq a/4$ and $E[|\#z - s|] \leq \sqrt{a/4}$.

We thus have obtained:

$$E[G(z')] \geq G(v^*) - \sqrt{a/4} \cdot 3a = G(v^*) - o(a^2).$$

□

If we denote by y^* the vector which we have arrived at after phase 1, and *result* the vector which is output by the algorithm, then we have:

- a) $F(y^*) = F(y_1^*, \dots, y_n^*) = F_a(y_1^*, \dots, y_a^*) = F_a(v^*)$.
- b) $F(\text{result}) = F(z'_1, \dots, z'_a, y_{a+1}^*, \dots, y_n^*) = F_a(z'_1, \dots, z'_a) = F_a(z')$.
- c) The number of clauses satisfied by *result* is $F(\text{result})$.

By c), $E[F(\text{result})]$ is the number which is of interest to us, and because of b), this is equal to $E[F_a(z')]$. By Lemma 2, we have

$$E[F_a(z')] \geq F_a(v^*) - o(a^2) = F(y^*) - o(a^2) \geq (3/4) \cdot \text{OPT}_{CC} - o(a^2).$$

It remains to be shown that the “error term” $o(a^2)$ is not too large compared to OPT_{CC} . This is what is done in the proof of the next theorem:

Theorem 1. *Algorithm “Randomized Rounding with Preprocessing” is a randomized polynomial-time approximation algorithm for the MAX-2-SATCC problem with an asymptotic worst-case ratio of $3/4$.*

Proof. Observe that the clauses $C_1, \dots, C_m$ given as an input to the algorithm must contain at least $\binom{a}{2}$ mixed clauses on the variables $x_1, \dots, x_a$. The reason for this is that the objective function F is the sum of the functions P_C , where C ranges over the input clauses. As we have pointed out earlier, P_C only contains a positive coefficient for $x_i x_j$ if C is a mixed clause on x_i and x_j . Since by Lemma 1, the second phase of the algorithm starts with an objective function F_a which for all $x_i, x_j \in \{x_1, \dots, x_a\}$ with $i \neq j$ has a coefficient $d_{\{i,j\}} \geq 1$, there must be at least $\binom{a}{2} = \Omega(a^2)$ mixed clauses in the beginning. We can now prove that $\text{OPT}_{CC} = \Omega(a^2)$ by showing that there is an assignment with T ones which satisfies $\Omega(a^2)$ mixed clauses.

For this purpose, we choose a vector $b = (b_1, \dots, b_n)$ according to the uniform distribution, from the set of all assignments with exactly T ones. We analyze the expected number of clauses which are satisfied by b : By linearity of expectation, it is enough to compute the probability that a single (mixed) clause, say $C = x_i \vee \bar{x}_j$, is satisfied. x_i is satisfied with probability T/n , $\bar{x}_j$ is satisfied with probability $(n - T)/n$. For any T , one of the two is at least $1/2$, hence C is satisfied with probability at least $1/2$, and the expected number of clauses satisfied is at least one-half of all mixed clauses. In particular, there must be a b which satisfies at least one-half of all mixed clauses. Since the given MAX-2-SATCC instance contains at least $\Omega(a^2)$ mixed clauses, we have that at least $\Omega(a^2)$ are satisfied by b .

This means that $OPT_{CC} = \Omega(a^2)$ and for the assignment *result* output by the algorithm, it holds that the expected number of clauses it satisfies is $E[F(\text{result})] \geq (3/4) \cdot OPT_{CC} - o(a^2) \geq (3/4) \cdot OPT_{CC} - o(OPT_{CC})$. $\square$

Conclusion and Outlook

The approach which we have presented might be interesting in its own right, it could be applicable in other situations where randomized rounding is involved.

Abstracting from the details, the greedy preprocessing step left us with a problem which in a certain sense is “dense”, i.e., where we are left with a problem on a variables, where for each pair of variables, there is one clause on those two variables in the input instance. Dense instances of problems are often easier to handle, e.g., for dense instances of the MAX- k -SAT problem, even polynomial-time approximation schemes (PTAS) are known, see the paper by Arora, Karger and Karpinski [AKK].

As far as MAX- k -SATCC for $k > 2$ is concerned, the following approach might be promising: First, apply a greedy preprocessing phase which leaves a dense instance. Then, apply the techniques from [AKK] for MAX- k -SAT to this instance. This might give approximation algorithms which have an approximation ratio better than $1 - (1/e)$ (tending to this value if k gets large, of course).

The reader familiar with the article [AKK] might wonder whether the results in there could perhaps be directly applied to the MAX-2-SATCC instance, but there is a clear sign that this is not the case: As we have mentioned before, the existence of a polynomial-time approximation algorithm for MAX-2-SAT with worst-case approximation ratio larger than $21/22$ would imply $P=NP$, so an analogous statement does hold for MAX-2-SATCC. On the other hand, [AKK] even yields PTASs, hence it should be clear that some sort of greedy preprocessing step is needed.

In this paper, we have not considered the weighted version of the MAX-2-SATCC problem. The reason for this is that the computations become a little bit more complicated and would obscure the main idea behind our algorithm.

Let us finish our remarks with an open problem. The MAXCUT problem, which is a special graph partitioning problem, is known to have a polynomial-time approximation algorithm with an approximation ratio of 0.878 [GW2]. Yet, its cardinality constrained variant – where the size of one of the parts is given in advance – can up to now not be approximated to a similar degree. The best known approximation ratio was achieved by Feige and Langberg [FL] who proved that with the help of semidefinite programming, an approximation ratio of $1/2 + \varepsilon$, for some $\varepsilon > 0$, can be obtained. The main message behind this result is that also for cardinality-constrained problems, semidefinite programming can lead to approximation ratios which are better than those known to be achievable by linear programming. The question remains whether it is possible to apply semidefinite programming to also obtain better approximation algorithms for the MAX-2-SATCC problem.

References

- [AKK] S. Arora, D. R. Karger and M. Karpinski, *Polynomial Time Approximation Schemes for Dense Instances of NP-Hard Problems*, J. Computer and System Sciences 58(1), 193–210, 1999.
- [AS] A. A. Ageev and M. Sviridenko, *Approximation Algorithms for Maximum Coverage and Max Cut with Given Sizes of Parts*, Proc. of the Seventh Conference on Integer Programming and Combinatorial Optimization (IPCO), 17–30, 1999.
- [AW] T. Asano and D. P. Williamson, *Improved Approximation Algorithms for MAX SAT*, J. Algorithms 42(1), 173–202, 2002.
- [BM] M. Bläser and B. Manthey, *Improved Approximation Algorithms for MAX-2SAT with Cardinality Constraints*, Proc. of the Int. Symp. on Algorithms and Computation (ISAAC), 187–198, 2002.
- [F] U. Feige, *A Threshold of $\ln n$ for Approximating Set Cover*, J. of the ACM 45(4), 634–652, 1998.
- [FL] U. Feige and M. Langberg, *Approximation Algorithms for Maximization Problems Arising in Graph Partitioning*, J. Algorithms 41(2), 174–211, 2001.
- [GW1] M. X. Goemans and D. P. Williamson, *New $3/4$ -Approximation Algorithms for the Maximum Satisfiability Problem*, SIAM J. Discrete Mathematics, 7(4), 656–666, 1994.
- [GW2] M. X. Goemans and D. P. Williamson, *Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming*, J. of the ACM 42(6), 1115–1145, 1995.
- [H] J. Håstad, *Some Optimal Inapproximability Results*, J. of the ACM 48(4), 798–859, 2001.
- [MR] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
- [S] M. Sviridenko, *Best Possible Approximation Algorithm for MAX SAT with Cardinality Constraint*, Algorithmica 30(3), 398–405, 2001.
- [V] V. V. Vazirani, *Approximation Algorithms*, Springer, 2001.

On-Demand Broadcasting Under Deadline

Bala Kalyanasundaram* and Mahe Velauthapillai**

Georgetown University
{kalyan,mahe}@cs.georgetown.edu

Abstract. In broadcast scheduling multiple users requesting the same information can be satisfied with one single broadcast. In this paper we study preemptive on-demand broadcast scheduling with deadlines on a single broadcast channel. We will show that the upper bound results in traditional real-time scheduling does not hold under broadcast scheduling model. We present two easy to implement online algorithms BCast and its variant BCast2. Under the assumption the requests are approximately of equal length (say k), we show that BCast is $O(k)$ competitive. We establish that this bound is tight by showing that every online algorithm is $\Omega(k)$ competitive even if all requests are of same length k . We then consider the case where the laxity of each request is proportional to its length. We show that BCast is constant competitive if all requests are approximately of equal length. We then establish that BCast2 is constant competitive for requests with arbitrary length. We also believe that a combinatorial lemma that we use to derive the bounds can be useful in other scheduling system where the deadlines are often changing (or advanced).

1 Introduction

On demand pay-per-view services have been on the increase ever since they were first introduced. In this model, there is a collection of documents such as news, sports, movies, etc., for the users to view. Typically, broadcasts of such documents are scheduled ahead of time and the users are forced to choose one of these predetermined times. Moreover, the collection of documents broadcasted on such regular basis tend to be small. Even though the collection could change dynamically (but slowly), this collection is considered to be the collection of "hot" documents by the server. Recently many companies, for example *TIVO*, *REAL*, *YESTV* have introduced true on-demand services where a user dynamically makes a request for a document from a large set of documents. This has the advantage of dealing with larger set of documents and possibly satisfying the true demand of the users. Generally, the service provider satisfies the request (if possible) for each user by transmitting the document independently for each user. This leads to severe inefficiencies since the service provider may repeat

* Supported in part by NSF under grant CCR-0098271, Airforce Grant, AFOSR F49620-02-1-0100 and Craves Family Professorship funds.

** Supported in part by a gift from AT&T and McBride Endowed Chair funds.

the same transmission many times. Broadcasting has the advantage of satisfying many users with the same request with one broadcast [1,3,6,9]. But, shifting from transmitting at a fixed time or regular intervals to true on-demand broadcasting has a major disadvantage. A user does not know whether the request will be satisfied or not and may experience a long wait. Even if we minimize the average response time (see [4]) for the user, unpredictability of the response time may be completely unacceptable for many users. It would be appropriate if the user assigns a deadline after which the completion of the request bears no value to the user.

In this paper we study preemptive on-demand broadcasting with deadline on a single broadcast channel. We associate an arrival time, a requested document, a deadline and a profit with each request. The system receives requests at the arrival time and knows nothing regarding future demands when it decides to broadcast a piece of a document. Whenever a request is satisfied on or before its deadline, the system earns the profit specified by the request. Otherwise, the system does not earn any profit from the request. This is often referred to as *soft deadline*. Our goal is to maximize the overall profit of the system.

First we consider the case where all the documents are approximately equal in length, which we call the $O(1)$ -length condition. This is motivated by the fact that most of the documents (e.g., movies) are about the same length. We present an easy to implement online algorithm which we call *BCast*. Then we prove that this algorithm is $O(k)$ competitive, where k is the length of the longest request. We also show that this result is tight by showing that every online algorithm is $\Omega(k)$ competitive. We then answer the following question: Under what condition can we find a constant competitive algorithm for this problem? We prove that *BCast* is constant competitive if laxity of each request is proportional to the length of the requested document (i.e., laxity assumption) and all documents are approximately of same length (i.e., length assumption). We then consider the case where the lengths of the requested documents differ arbitrarily. Does there exist an online algorithm with constant competitive ratio for this case? We answer the question by modifying *BCast* to handle arbitrary lengths. We prove that the modified algorithm, we call it *BCast2*, is constant competitive under *laxity assumption*. We also compare and contrast previous results in real-time scheduling with deadline [1,12].

1.1 Definitions and Model

We assume that users request for document from a collection $\{m_1, m_2, \dots\}$. This collection could be dynamically changing since our upper bounds are independent of the number of documents in the collection. A **document** m_i has ℓ_i indivisible or non-preemptable segments or chapters. We say that ℓ_i is the length of the document m_i and it does not vary over time. We assume that segments are approximately identical in size so that exactly one segment of any document can be broadcasted at a time on a single channel.

With respect to any document, we assume that the broadcast schedule is cyclical in nature. That is, if a document has 4 segments, (namely 1,2,3 and 4) then the i th broadcast of the document will be segment $(i - 1) \bmod 4 + 1$.

We assume that users request only entire documents. The *length of the request* is nothing but the length of the requested document. Moreover, users can assemble a document m_i if they receive all of the ℓ_i segments in any of the ℓ_i cyclical orders. Further, schedule on a single channel may choose different documents on consecutive time units as long as cyclical schedule is maintained with respect to each document. It is not hard to establish that noncyclic broadcast does not benefit the system if partial document is of no use to the individual users. See [1] for more details about on-demand broadcasting with deadlines.

In this paper, we deal with single channel broadcast scheduling. But, when we establish lower bounds, we show that even multiple channels or multiple broadcast per unit time does not provide significant benefit to the online algorithm. In order to establish such lower bound results, we introduce the following definitions.

We say that an algorithm is *s-speed* algorithm, if the algorithm is allowed to schedule s broadcasts for each time unit. For $s > 1$, more than one broadcast of a document at any time is possible.

We say that an algorithm is *m-channel* algorithm, if the algorithm is allowed to schedule broadcasts of m different documents at each time. Multiple broadcast of the same document is not allowed at any time.

Finally, we give a natural extension (to broadcast scheduling) of two standard algorithms from traditional real-time scheduling. Ties are broken arbitrarily.

Earliest Deadline First (EDF): At each broadcasting step, among all documents, **EDF** selects the one that has a pending satisfiable request with earliest deadline.

Least Laxity First (LLF): At each broadcasting step, among all documents, **LLF** selects the one that has a pending satisfiable request with least laxity.

The problem we consider in this paper is online in nature. The request for documents are presented to the system at the arrival time. A request R_i is a four tuple $(r_i, d_i, m_{z(i)}, p_i)$ which consists of an arrival time r_i , a deadline d_i , a requesting document $m_{z(i)}$ and payment p_i . The length of the request is $\ell_{z(i)}$. The use of $z(i)$ is to indicate that the request R_i does not always deal with document i .

The deadline specified in a request is a *soft deadline*. It means that the system gets paid p_i if the request is satisfied by the deadline d_i . But failure to satisfy R_i by its deadline does not bring any catastrophic consequence other than the loss of potential pay p_i to the system. Our objective is to maximize the revenue for the system.

Suppose I be the input given to *s-speed* online algorithm A . Let $C \subseteq I$ be the set of inputs satisfied by A by their deadline. We use the notation $A_s(I)$ to denote the $\sum_{R_i \in C} p_i$, the total profit earned by *s-speed* algorithm A on input I . We also use the notation $\text{OPT}(I)$ to denote the maximum profit that an offline optimal *1-speed* algorithm can earn.

An algorithm A is said to be a s -speed c -approximation algorithm if

$$\max_{\text{Inputs } I} \frac{A_s(I)}{\text{OPT}(I)} \leq c.$$

An algorithm A is said to be c -competitive, or said to have *competitive ratio* c , if A is a 1-speed c -approximation algorithm.

Request Pay-off Density Δ_i : This quantity for a request $R_i=(r_i, d_i, m_{z(i)}, p_i)$ is denoted by Δ_i and is defined to be $p_i/\ell_{z(i)}$.

For constants $\epsilon > 0$ and $c \geq 1$, we say that a set of requests I and the set of documents $\{m_1, m_2, \dots\}$, satisfy

a. **ϵ -laxity condition**, if for all requests $R_i \in I$, $d_i - r_i \geq (1 + \epsilon)\ell_{z(i)}$.

b. **c -length condition** if for all pairs of documents m_i and m_j , we have $\ell_i/\ell_j \leq c$.

The following two definitions are based on the **online algorithm** and the set of requests I under consideration. For ease of notation we will not indicate the online algorithm under consideration in the notation. It will be very clear from the context, since we only consider two different online algorithms and they are in two different sections.

Set of Live Requests $L_i(t)$: A request $R_i=(r_i, d_i, m_{z(i)}, p_i)$ is live at time t if the request has not been completed at time t and has a chance of being completed if the algorithm were to broadcast $m_{z(i)}$ exclusively from time t until its deadline. That is, $(d_i - t) \geq (\ell_{z(i)} - b)$ where $b \geq 0$ is the number of broadcasts of document $m_{z(i)}$ during the interval $[r_i, t)$. Given I , let $L_j(t)$ be the set of live requests for the document m_j at time t .

Document Pay-off Density $M_i(t)$: It is the sum of all the pay-off densities of the live-request pending for the document at time t . $M_i(t) = \sum_{R_j \in L_i(t)} \Delta_j$

1.2 Previous Results and Our Results

Broadcast scheduling problem has been studied previously by [1,3,6,5,9,10]. Most of the results consider average response time for the users. In these papers, there is no deadline associated with each request. Every request is eventually satisfied. But, each user experiences a response time equal to time-of-completion minus time-of-request. First, we [9] showed that there is an offline 3-speed 3-approximation for this problem using LP-based techniques. Later Gandhi et.al [6,7] improved the bounds for this offline case. Recently, Edmonds et. al [4] developed $O(1)$ -speed $O(1)$ -approximation online algorithm for the average response time case. They proved it by showing how to convert online algorithm from traditional scheduling domain to broadcasting domain. Our paper differs fundamentally from all of the previous work in broadcast scheduling. Independent to our work, Kim et. al [11] obtained constant competitive algorithm for the broadcasting problem with deadline when $O(1)$ -length condition is satisfied.

In section 2 we prove lower bound results. We first consider soft deadline case where the objective function is to maximize the overall profit. We prove that the competitive ratio of every deterministic online algorithm is $\Omega(k)$ (where k is

the length of the longest request) for the on-demand broadcasting problem with deadlines and preemption. Then we show that the competitive ratio does not improve significantly even if we allow m simultaneous broadcast of different documents at each time step for the online algorithm while offline optimal broadcasts only once. In this case we show a lower bound of $\Omega(k/m)$ on the competitive ratio.

Next we consider hard deadline case where we must satisfy each and every request. We consider only those set of requests I , such that there exists a schedule that broadcasts at most once each time, and satisfy all the requests in I . In the traditional single processor real-time scheduling, it is well known that LLF and EDF produces such schedule. For the single channel broadcast scheduling problem, we prove that even s -speed LLF and EDF algorithms do not satisfy every request even if 1-speed optimal satisfy all. Further, we show that there is no 1-speed online algorithm that can finish all the requests, even if 1-speed optimal satisfy all.

In section 3 we prove upper bound results. We do this by defining two algorithms BCast and BCast2. We first prove that BCast is $O(kc)$ competitive where k is the length of the longest request and c is the ratio of the length of the longest to the shortest request. As a corollary, if the set of documents satisfy $O(1)$ -length condition, then BCast is $O(k)$ competitive. We then show that BCast is constant competitive if the set of requests and the set of documents satisfy both $O(1)$ -length condition and $O(1)$ -laxity condition. We then modify BCast, which we call BCast2, in order to relax the $O(1)$ -length condition. We prove that BCast2 is $O(1)$ competitive if $O(1)$ -laxity condition alone is satisfied. Due to page limitations proofs of many theorems and lemmas have been omitted.

2 Lower Bound Results

In this section we prove lower bound results on broadcast scheduling with deadlines. We also compare these lower bound results with some of the lower and upper bound results in traditional (non-broadcasting setup) real-time scheduling.

2.1 Soft Deadlines

Recall that there is a simple constant competitive algorithm for traditional real-time scheduling with soft deadlines if all jobs are approximately of the same length [8]. In contrast, we show that it is not the case in broadcast scheduling under soft deadline.

Theorem 1. *Suppose all the documents are of same length k . Then every deterministic online algorithm is $\Omega(k)$ competitive for the on-demand broadcasting problem with deadlines and preemption.*

Proof. (of Theorem 1) Let $k > 0$ and A be any deterministic online algorithm. The adversary uses $k + 1$ different documents. The length of each document is k and the payoff for each request is 1. We will construct a sequence of requests such that A is able to complete only one request while the offline completes k requests. The proof proceeds in time steps. At time 0, $k + 1$ requests for $k + 1$ different documents arrive. That is, $0 \leq i \leq k$, $R_i = (0, k, m_i, 1)$. WLOG, A broadcasts m_0 during the interval $[0, 1]$. For time $1 \leq t \leq k - 1$, let $A(t)$ be the document that A broadcasts during the interval $[t, t + 1]$. Adversary then issues k requests for k different documents other than $A(t)$ at time t where each request has zero laxity. Since each request has zero laxity, A can complete only one request. Since there are $k + 1$ different documents and A can switch broadcast at most k times during $[0, k]$, there is a document with k requests which the offline optimal satisfies. ■

In the proof of the above theorem, the offline optimal satisfied k requests out of $\Theta(k^2)$ possible requests and A satisfied one request. In the next section we will study the performance of some well known online algorithms assuming the offline algorithm must completely satisfy all the requests.

We now show that no online algorithm performs well even if online algorithm is allowed m broadcasts per unit time while offline optimal performs one broadcast per unit time.

Theorem 2. *Suppose all the documents are of same length k . For $m > 0$, every m -broadcast deterministic online algorithm is $\Omega(k/m)$ competitive for the on-demand broadcasting problem with deadlines and preemption.*

2.2 Hard Deadlines

In this subsection, we consider the input instance where offline optimal completes **all** the requests before their deadline. Recall that in the traditional single processor real-time scheduling, it is well known that LLF and EDF are optimal. However, we show that LLF and EDF perform poorly for broadcast scheduling even if we assume that they have s -speed broadcasting capabilities.

Theorem 3. *Let s be any positive integer. There exists a sequence of requests that is fully satisfied by the optimal (offline) algorithm, but not by s -speed EDF. There exists another sequence of requests that is fully satisfied by the optimal (offline) algorithm, but not by s -speed LLF.*

Recall that the proof of Theorem 1 uses $\Theta(k^2)$ requests where the optimal offline can finish $\Theta(k)$ requests to establish a lower bound for online algorithm. The following theorem shows that no online algorithm can correctly identify a schedule to satisfy each and every request if one such schedule exists.

Theorem 4. *Let A be any online algorithm. Then there exists a sequence of requests that is satisfied by the optimal (offline) algorithm, but not by A .*

3 Upper Bound

Before we describe our algorithms and their analysis, we give intuitive reasoning to the two assumptions (length and laxity) as well as their role in the analysis of the algorithm.

When an online algorithm schedules broadcasts, it is possible that a request is partially satisfied before its deadline is reached. Suppose each user is willing to pay proportional to the length of the document he/she receives. Let us call it partial pay-off. On the contrary, we are interested actual pay-off which occurs only when the request is fully satisfied. Obviously, partial pay-off is at least equal to actual pay-off.

Definition 1. Let $0 < \alpha \leq 1$ be some constant. We say that an algorithm for the broadcast scheduling problem is α -greedy, if at any time the pay-off density of the chosen document of the algorithm is at least α times the pay-off density of any other document.

Our algorithms are α -greedy for some α . Using this greedy property and applying $O(1)$ -length property, we will argue that actual pay-off is at least a constant fraction of partial pay-off. Then applying $O(1)$ -laxity property, we will argue that the partial pay-off defined above is at least a fraction of the pay-off received by the offline optimal.

3.1 Approximately Same Length Documents

In this subsection we assume that the length of the requests are approximately within a constant factor of each other, which we call $O(1)$ -length condition. We first present a simple algorithm that we call *BCast*. We prove that the competitive ratio of this algorithm is $O(k)$ where k is the length of the longest request, thus matching the lower bound shown in Theorem 1. We then show that if in addition to $O(1)$ -length condition $O(1)$ -laxity condition is also satisfied then *BCast* is constant competitive.

BCast: At each time step, the algorithm broadcasts a chapter of a document. We will now describe what document the algorithm chooses at each time step. With respect to any particular document, the algorithm broadcasts chapters in the cyclical wrap-around fashion. In order to do so, the algorithm maintains the next chapter that it plans to transmit to continue the cyclical broadcast. The following description deals with the selection of document at each time step.

1. **At time 0**, choose the document m_i with the highest $M_i(0)$ (document pay-off density) to broadcast.
2. **At time t**
 - a) Compute $M_i(t)$'s for each document and let m_j be the document with highest pay-off density $M_j(t)$.
 - b) Let m_c be the last transmitted document. If $M_j(t) \geq 2M_c(t)$ then transmit m_j . Otherwise continue transmitting m_c .

End BCast

Observation 1 *BCast is $\frac{1}{2}$ -greedy for the broadcast scheduling problem. On the negative side, it is quite possible that BCast never satisfy even a single request. This happens when there are infinitely many requests such that the pay-off density of some document is exponentially approaching infinity. So, we assume that the number of requests is finite.*

Definition 2. 1. For ease of notation, we use A to denote online algorithm BCast.

2. Let $m_{A(t)}$ be the document transmitted by algorithm A (i.e., BCast) at time t . For ease of presentation, we abuse the notation and say that $A(t)$ is the document transmitted by A at time t .

3. Let t_0 be the starting time, $t_1, \dots, t_N$ be the times at which BCast changed documents for broadcast and t_{N+1} be the time at which BCast terminates.

4. For $0 \leq i \leq N-1$, let C_i be the set of all requests completed by BCast during the interval $[t_i, t_{i+1})$.

5. C_N be the set of all requests completed by BCast during the interval $[t_N, t_{N+1}]$.

6. $C = \cup_{i=0}^N C_i$.

Next we will proceed to show that the algorithm BCast is $O(k)$ competitive. First we prove some preliminary lemmas.

Lemma 1. For any $0 \leq i \leq N$, $M_{A(t_i)}(t_i) \leq M_{A(t_i)}(t_{i+1}) + \sum_{R_j \in C_i} \Delta_j$.

Lemma 2. Let k be the length of the longest document. $\sum_{t \in [t_i, t_{i+1})} M_{A(t)}(t) \leq k M_{A(t_i)}(t_{i+1}) + \sum_{R_j \in C_i} p_j$.

Lemma 3. $\sum_{i=0}^N M_{A(t_i)}(t_{i+1}) \leq 2 \sum_{R_j \in C} \Delta_j$.

Proof. (of Lemma 3)

We prove this by a point distribution argument. Whenever a request R_j is completed by BCast during the time interval $[t_i, t_{i+1})$, we will give $2\Delta_j$ points to R_j . Observe that total points that we gave is equal to the right hand side of the equation in the lemma. We will now partition the points using a redistribution scheme into $N+1$ partitions such that the i th partition receives at least $M_{A(t_i)}(t_{i+1})$. The lemma then follows.

All partitions initially have 0 points. Our distribution process has $N+1$ iterations where at the end of i iteration, $N+2-i$ th partition will receive $2M_{A(t_{N+1-i})}(t_{N+2-i})$ points. During the $i+1$ st iteration $N+2-i$ th partition will donate $M_{A(t_{N+1-i})}(t_{N+2-i})$ points to $N+1-i$ th partition. Also, $2\Delta_j$ points given each R_j completed during the interval $[t_{N+1-i}, t_{N+2-i}]$ is also given to $N+1-i$ th partition.

We argue that $N+1-i$ th partition receives $2M_{A(t_{N-i})}(t_{N+1-i})$. At time t_{N+1-i} , BCast jumps to a new document. So, $2M_{A(t_{N-i})}(t_{N+1-i}) \leq M_{A(t_{N+1-i})}(t_{N+1-i})$. Apply lemma 1, we have $M_{A(t_{N+1-i})}(t_{N+1-i}) \leq \sum_{R_j \in C_{N+1-i}} \Delta_j + M_{A(t_{N+1-i})}(t_{N+2-i})$. Combining these two inequalities we get, $2M_{A(t_{N-i})}(t_{N+1-i}) \leq \sum_{R_j \in C_{N+1-i}} \Delta_j + M_{A(t_{N+1-i})}(t_{N+2-i})$. The result then follows. ■

Lemma 4. Let k be the maximum length of any request. $\sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \leq k \sum_{i=0}^N M_{A(t_i)}(t_{i+1}) + \sum_{R_j \in C} p_j$.

Lemma 5. Let c be the constant representing the ratio of the length of longest document to the length of shortest document. $\sum_{R_i \in C} p_i \geq \frac{1}{2c+1} \sum_{t=0}^{t_N} M_{A(t)}(t)$.

Proof. (of Lemma 5) By using Lemma 3 and Lemma 4 we get $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq 2k \sum_{R_j \in C} \Delta_j + \sum_{R_j \in C} p_j$. That is, $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq 2 \sum_{R_j \in C} k \Delta_j + \sum_{R_j \in C} p_j$. By definition of Δ_j $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq 2 \sum_{R_j \in C} k(p_j/\ell_j) + \sum_{R_j \in C} p_j$. Since c is the ratio of the length of longest document to the length of shortest document, $\sum_{t=0}^{t_N} M_{A(t)}(t) \leq (2c+1) \sum_{R_j \in C} p_j$. ■

Lemma 6. Let C , OPT be the requests completed by BCast and offline optimal respectively. Then, $2k \sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq \sum_{R_j \in OPT} p_j - \sum_{R_j \in C} p_j$.

Proof. (of Lemma 6) For a moment imagine that offline optimal gets paid p_j/ℓ_j only for the first received chapter for each request $R_j \in OPT - C$. Let $FO(t)$ be the set of requests in OPT that receive their first broadcast at time t based on the schedule opt . Let $FOPT(t)$ be the sum of pay-off densities of the requests in $FO(t)$. Observe that $\sum_{t=0}^{t_{N+1}} FOPT(t) \geq \sum_{R_j \in (OPT-C)} \Delta_j$ and $\sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq 1/2 \sum_{t=0}^{t_{N+1}} FOPT(t)$. Combining the above two inequalities, $\sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq 1/2 \sum_{R_j \in (OPT-C)} \Delta_j$. Multiplying by k and expanding the right hand side we get, $2k \sum_{t=0}^{t_{N+1}} M_{A(t)}(t) \geq \sum_{R_j \in OPT} p_j - \sum_{R_j \in C} p_j$. ■

Theorem 5. Algorithm BCast is $O(kc)$ competitive where k is the length of the longest request and c is the ratio of the length of the longest to the shortest document.

Proof. (of Theorem 5) From Lemma 5 $2k(2c+1) \sum_{R_i \in C} p_i \geq 2k \sum_{t=0}^{t_{N+1}} M_{A(t)}(t)$. From Lemma 6, $2k(2c+1) \sum_{R_i \in C} p_i \geq \sum_{R_i \in OPT} p_i - \sum_{R_i \in C} p_i$. Simplyfying, $[2k(2c+1) + 1] \sum_{R_i \in C} p_i \geq \sum_{R_i \in OPT} p_i$. ■

Corollary 1. BCast is $O(k)$ competitive if requests are approximately same length.

Next we will prove that the BCast algorithm is $O(1)$ competitive if the laxity is proportional to length.

For ease of presentation, we use the notation opt to represent the offline optimal algorithm and OPT be the set of requests satisfied by opt . First, we prove a key lemma that we use to derive upper bounds for two algorithms. Intuitively, each request in OPT is reduced in length to a small fraction of its original length. After reducing the length of each request, we advance the deadline of each request R_i to some time before $d_i - (1+\eta)\ell_{z(i)}$. We then show that there exists a pair of schedules S_1 and S_2 such that their union satisfy these reduced requests before their new deadline. Since a fraction of each request in OPT is scheduled, the partial pay-off is proportional to the total pay-off earned by the offline optimal schedule. We then argue that our greedy algorithm does better than both S_1 and S_2 . We think that this lemma may have applications in other areas of scheduling where one deals with sudden changes in deadlines.

Lemma 7. *Suppose $\delta = 2\epsilon/9$ and $\epsilon < 1/2$. Under ϵ -laxity assumption, there exists two schedules S_1 and S_2 such that the following property holds: For all $R_i \in OPT$, the number of broadcasts of document m_i in both S_1 and S_2 during the interval $[r_i, d_i - (1 + \delta + \epsilon/2)\ell_i]$ is at least $\delta\ell_i$.*

In the following lemma, we establish the fact that the partial pay-off for A (i.e., $BCast$) is at least a constant fraction of the pay-off earned by offline optimal algorithm opt when $O(1)$ -laxity condition is met.

Lemma 8. *Under the ϵ -laxity assumption and for some $\gamma > 0$ the following holds. $\sum_t M_{A(t)}(t) \geq \gamma \sum_{R_i \in OPT} p_i$.*

Theorem 6. *Under both $O(1)$ -length and ϵ -laxity conditions, the algorithm $BCast$ is $O(1)$ competitive.*

3.2 Arbitrary Length Documents

In this subsection, we consider the case where the length of the document vary arbitrarily. However, we continue to assume that ϵ -laxity condition is satisfied.

We will present a modified online algorithm, which we call $BCast2$, and prove that it is $O(1)$ competitive under ϵ -laxity condition.

Before we proceed to modify $BCast$, we point out the mistake that $BCast$ makes while dealing with arbitrary length documents. When $BCast$ jumps from one document (say m_i) to another (say m_j) at time t , it does so based only on the density of the documents and bluntly ignores their length. At time t , we have $M_j(t) \geq 2M_i(t)$. But at time $t+1$, it could be the case that $M_j(t+1)$ gone down to a level such that $M_j(t+1)$ is just greater than $\frac{1}{2}M_i(t+1)$. However, this does not trigger the algorithm to switch back to document m_i from m_j . As a consequence, for long documents such as m_i , we will accumulate lots of partially completed requests and thus foil our attempt to show that the total cost earned by completing requests is not a constant fraction of partial pay-off (i.e., accumulated pay-off if even partially completed requests pay according to the percentage of completion).

In order to take care of this situation, our new algorithm $BCast2$ maintains a stack of previously transmitted document. Now switching from one document to another is based on the result of checking two conditions. First, make sure that the density of the document on top of the stack is still a small fraction of the density of the transmitting document. This is called *condition 1* in the algorithm. Second, make sure that there is no other document with very high density. This is called *condition 2* in the algorithm. If any one or both these conditions are violated then the algorithm will switch to a new document to broadcast. In order to make this idea clear (and make it work), we introduce two additional labeling on the requests. As before, these definitions critically depends on the algorithm under consideration.

Startable Request: We say that a request R_i is startable at time t , if the algorithm has not broadcasted document m_i during $[r_i, t]$ and $r_i \leq t \leq d_i - (1 + \epsilon/2)\ell_i$.

Started Request: We say that a request R_i is started at time t if it is live at time t and broadcast of document m_i took place in the interval $[r_i, d_i - (1+\epsilon/2)\ell_i]$. Observe that the document pay-off density is redefined to be based on the union of started and startable requests as opposed to live requests.

$M_k(t)$ denote the sum of the densities of the started or startable request at time t for the document m_k .

$SM_k(t)$ denote the sum of the densities of the started request at time t for the document m_k .

TM_k denote the density of document m_k at the time of entry into the stack (T stands for the threshold value). As long as the document m_k stays on the stack, this value TM_k does not change.

BCast2 is executed by the service providers. Assume the service provider has n distinct documents. The algorithm maintains a stack; each item in the stack has the following two information:

- 1) Document name say m_k .
- 2) Started density value $SM_k(t)$ of the document at the time t it goes on the stack. We refer it TM_k for document m_k and it is time independent.

Initially stack is empty.

BCast2: c_1 and α are some positive constants that we will fix later.

1. At time $t = 0$ choose the document with the highest M_i (document pay-off density) and transmit:
2. **For** $t = 1, 2 \dots$
 - a) Let m_j be the document with the highest $M_j(t)$ value.
 - b) Let m_k be the document on top of the stack (m_k is undefined if the stack is empty).
 - c) Let m_i be the document that was broadcast in the previous time step.
 - d) **While** $((SM_k(t) \leq \frac{1}{2}TM_k)$ and Stack Not Empty)
 - e) pop stack.
 - f) Now m_k be the document on top of stack.
 - g) Condition 1. $SM_k(t) \geq \frac{c_1\epsilon}{(1+\alpha)}M_i(t)$
 - h) Condition 2. $M_j(t) \geq \frac{2(1+\alpha)}{c_1\epsilon}M_i(t)$
 - i) If both conditions are false continue broadcasting document m_i .
 - j) If condition (1) is true then broadcast m_k , pop m_k from the stack (do not push m_i on the stack).
 - k) If condition (2) is true the push m_i on the stack along with the value $M_i(t)$ (which is denoted by TM_i), broadcast m_j .
 - l) If both conditions are true then choose m_j to broadcast only if $M_j(t) \geq \frac{2(1+\alpha)}{c_1\alpha}M_k(t)$. Otherwise broadcast m_k , pop m_k from the stack (do not push m_i on the stack). We will later establish the fact that both conditions 1 and 2 are false for the current choice of broadcast m_k .
3. **EndFor**

End BCast2

For ease of presentation we overload the term *B*Cast2 to represent the set of all requests completed by BCast2 before their deadline. As before, we use A to denote algorithm BCast2 in our notation. Without the $O(1)$ -length condition, we will now establish the fact that the total pay-off for completed requests for BCast2 is proportional to the partial pay-off where every request pays proportional to the percentage of completion.

Lemma 9. For $c_1 \leq \frac{3}{32}$, $\sum_{R_j \in BCast2} b_j \geq \frac{\alpha}{2(1+\alpha)} \sum_t M_{A(t)}(t)$

Theorem 7. Assuming ϵ -laxity condition, *B*Cast2 is constant competitive algorithm for the broadcast scheduling problem.

References

1. A. Acharya and S. Muthukrishnan. Scheduling On-demand Broadcasts: New Metrics and Algorithms. In *MobiCom*, 1998.
2. A. Bar-Noy, S. Guha, y. Katz, and J. Naor. Throughput maximization of real-time scheduling with batching. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms*, January 2002.
3. Y. Bartal and S. Muthukrishnan. Minimizing Maximum Response Time in Scheduling Broadcasts. In *SODA*, pages 558–559, 2000.
4. J. Edmonds and K. Pruhs. Multicast pull scheduling: When fairness is fine. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms*, January 2002.
5. T. Erlebach and A. Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *Proceedings of ACM/SIAM Symposium on Discrete Algorithms*, January 2002.
6. R. Gandhi, S. Khuller, Y. Kim, and Y-C Wan. Approximation algorithms for broadcast scheduling. In *Proceedings of Conference on Integer Programming and Combinatorial Optimization*, 2002.
7. R. Gandhi, S. Khuller, S. Parthasarathy, and S. Srinivasan. Dependent rounding in bipartite graphs. *IEEE Symposium on Foundations of Computer Science*, 2002.
8. B. Kalyanasundaram and K.R. Pruhs. Speed is as Powerful as Clairvoyance. *IEEE Symposium on Foundation of Computation*, pages 214–221, 1995.
9. B. Kalyanasundaram, K.R. Pruhs, and M. Velauthapillai. Scheduling Broadcasts in Wireless Networks. *Journal of Scheduling*, 4:339–354, 2001.
10. C. Kenyon, N. Schabanel, and N Young. Polynomial-time approximation schemes for data broadcasts. In *Proceedings of Symposium on Theory of Computing*, pages 659–666, 2000.
11. J. H. Kim and K. Y. Chowa. Scheduling broadcasts with deadlines. In *COCOON*, 2003.
12. C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *ACM Symposium on Theory of Computing*, pages 140–149, 1997.

Improved Bounds for Finger Search on a RAM^{*}

Alexis Kaporis, Christos Makris, Spyros Sioutas, Athanasios Tsakalidis,
Kostas Tsihlas, and Christos Zaroliagis

Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
and Department of Computer Engineering and Informatics,
University of Patras, 26500 Patras, Greece
{kaporis,makri,sioutas,tsak,tsihlas,zaro}@ceid.upatras.gr

Abstract. We present a new finger search tree with $O(1)$ worst-case update time and $O(\log \log d)$ expected search time with high probability in the Random Access Machine (RAM) model of computation for a large class of input distributions. The parameter d represents the number of elements (distance) between the search element and an element pointed to by a finger, in a finger search tree that stores n elements. For the need of the analysis we model the updates by a “balls and bins” combinatorial game that is interesting in its own right as it involves insertions and deletions of balls according to an unknown distribution.

1 Introduction

Search trees and in particular finger search trees are fundamental data structures that have been extensively studied and used, and encompass a vast number of applications (see e.g., [12]). A *finger search tree* is a leaf-oriented search tree storing n elements, in which the search procedure can start from an arbitrary element pointed to by a *finger* f (for simplicity, we shall not distinguish throughout the paper between f and the element pointed to by f). The goal is: (i) to find another element x stored in the tree in a time complexity that is a function of the “distance” (number of leaves) d between f and x ; and (ii) to update the data structure after the deletion of f or after the insertion of a new element next to f . Several results for finger search trees have been achieved on the Pointer Machine (PM) and the Random Access Machine (RAM) models of computation.

In this paper we concentrate on the RAM model. W.r.t. worst-case complexity, finger search trees with $O(1)$ update time and $O(\log d)$ search time have already been devised by Dietz and Raman [5]. Recently, Andersson and Thorup [2] improved the search time to $O(\sqrt{\log d / \log \log d})$, which is optimal since there exists a matching lower bound for searching on a RAM. Hence, there is no room for improvement w.r.t. the worst-case complexity.

^{*} This work was partially supported by the IST Programme of EU under contract no. IST-1999-14186 (ALCOM-FT), by the Human Potential Programme of EU under contract no. HPRN-CT-1999-00104 (AMORE), and by the Carathéodory project of the University of Patras.

However, simpler data structures and/or improvements regarding the search complexities can be obtained if randomization is allowed, or if certain classes of input distributions are considered. A notorious example for the latter is the method of *interpolation search*, first suggested by Peterson [16], which for random data generated according to the *uniform* distribution achieves $\Theta(\log \log n)$ expected search time. This was shown in [7,15,19]. Willard in [17] showed that this time bound holds for an extended class of distributions, called *regular*¹.

A natural extension is to adapt interpolation search into dynamic data structures, that is, data structures which support insertion and deletion of elements in addition to interpolation search. Their study was started with the works of [6, 8] for insertions and deletions performed according to the uniform distribution, and continued by Mehlhorn and Tsakalidis [13], and Andersson and Mattsson [1] for μ -random insertions and *random* deletions, where μ is a so-called *smooth* density. An insertion is μ -random if the key to be inserted is drawn randomly with density function μ ; a deletion is random if every key present in the data structure is equally likely to be deleted (these notions of randomness are also described in [10]).

The notion of *smooth* input distributions that determine insertions of elements in the update sequence were introduced in [13], and were further generalized and refined in [1]. Given two functions f_1 and f_2 , a density function $\mu = \mu[a, b](x)$ is (f_1, f_2) -*smooth* [1] if there exists a constant β , such that for all c_1, c_2, c_3 , $a \leq c_1 < c_2 < c_3 \leq b$, and all integers n , it holds that

$$\int_{c_2 - \frac{c_3 - c_1}{f_1(n)}}^{c_2} \mu[c_1, c_3](x) dx \leq \frac{\beta \cdot f_2(n)}{n}$$

where $\mu[c_1, c_3](x) = 0$ for $x < c_1$ or $x > c_3$, and $\mu[c_1, c_3](x) = \mu(x)/p$ for $c_1 \leq x \leq c_3$ where $p = \int_{c_1}^{c_3} \mu(x) dx$. The class of smooth distributions is a superset of both regular and uniform classes.

In [13] a dynamic interpolation search data structure was introduced, called *Interpolation Search Tree (IST)*. This data structure requires $O(n)$ space for storing n elements. The amortized insertion and deletion cost is $O(\log n)$, while the expected amortized insertion and deletion cost is $O(\log \log n)$. The worst case search time is $O(\log^2 n)$, while the expected search time is $O(\log \log n)$ on sets generated by μ -random insertions and random deletions, where μ is a $(\lceil n^a \rceil, \sqrt{n})$ -smooth density function and $\frac{1}{2} \leq a < 1$. An IST is a multi-way tree, where the degree of a node u depends on the number of leaves of the subtree rooted at u (in the ideal case the degree of u is the square root of this number). Each node of the tree is associated with two arrays: a REP array which stores a set of sample elements, one element from each subtree, and an ID array that stores a set of sample elements approximating the inverse distribution function. The search algorithm for the IST uses the ID array in each visited node to interpolate REP and locate the element, and consequently the subtree where the search is to be continued.

¹ A density μ is regular if there are constants b_1, b_2, b_3, b_4 such that $\mu(x) = 0$ for $x < b_1$ or $x > b_2$, and $\mu(x) \geq b_3 > 0$ and $|\mu'(x)| \leq b_4$ for $b_1 \leq x \leq b_2$.

In [1], Andersson and Mattsson explored further the idea of dynamic interpolation search by observing that: (i) the larger the ID array the bigger becomes the class of input distributions that can be efficiently handled with an IST-like construction; and (ii) the IST update algorithms may be simplified by the use of a static, implicit search tree whose leaves are associated with binary search trees and by applying the incremental global rebuilding technique of [14]. The resulting new data structure in [1] is called the *Augmented Sampled Forest (ASF)*. Assuming that $H(n)$ is an increasing function denoting the height of the static implicit tree, Andersson and Mattsson [1] showed that an expected search and update time of $\Theta(H(n))$ can be achieved for μ -random insertions and random deletions where μ is $(n \cdot g(H(n)), H^{-1}(H(n) - 1))$ -smooth and g is a function satisfying $\sum_{i=1}^{\infty} g(i) = \Theta(1)$. In particular, for $H(n) = \Theta(\log \log n)$ and $g(x) = x^{-(1+\varepsilon)}$ ($\varepsilon > 0$), they get $\Theta(\log \log n)$ expected search and update time for any $(n/(\log \log n)^{1+\varepsilon}, n^{1-\delta})$ -smooth density, where $\varepsilon > 0$ and $0 < \delta < 1$ (note that $(\lceil n^a \rceil, \sqrt{n})$ -smooth $\subset (n/(\log \log n)^{1+\varepsilon}, n^{1-\delta})$ -smooth). The worst-case search and update time is $O(\log n)$, while the worst-case update time can be reduced to $O(1)$ if the update position is given by a finger. Moreover, for several but more restricted than the above smooth densities they can achieve $o(\log \log n)$ expected search and update time complexities; in particular, for the uniform and any bounded distribution the expected search and update time becomes $O(1)$. The above are the best results so far in both the realm of dynamic interpolation structures and the realm of dynamic search tree data structures for μ -random insertions and random deletions on the RAM model.

Based upon dynamic interpolation search, we present in this paper a new finger search tree which, for μ -random insertions and random deletions, achieves $O(1)$ worst-case update time and $O(\log \log d)$ expected search time *with high probability* (w.h.p.) in the RAM model of computation for the same class of smooth density functions μ considered in [1] (Sections 3 and 4), thus improving upon the dynamic search structure of Andersson and Mattsson with respect to the expected search time complexity. Moreover, for the same classes of restricted smooth densities considered in [1], we can achieve $o(\log \log d)$ expected search and update time complexities w.h.p. (e.g., $O(1)$ times for the uniform and any bounded distribution). We would like to mention that the expected bounds in [1,13] have not been proved to hold w.h.p. Our worst-case search time is $O(\sqrt{\log d / \log \log d})$. To the best of our knowledge, this is the first work that uses the dynamic interpolation search paradigm in the framework of finger search trees.

Our data structure is based on a rather simple idea. It consists of two levels: the top level is a tree structure, called *static interpolation search tree* (cf. Section 2) which is similar to the static implicit tree used in [1], while the bottom level consists of a family of buckets. Each bucket is implemented by using the fusion tree technique [18]. However, it is not at all obvious how a combination of these data structures can give better bounds, since deletions of elements may create chains of empty buckets. To alleviate this problem and prove the expected search bound, we use an idea of independent interest. We model the insertions and dele-

tions as a combinatorial game of bins and balls (Section 5). This combinatorial game is innovative in the sense that it is not used in a load-balancing context, but it is used to model the behaviour of a dynamic data structure as the one we describe in this paper. We provide upper and lower bounds on the number of elements in a bucket and show that, w.h.p., a bucket never gets empty. This fact implies that w.h.p. there cannot exist chains of empty buckets, which in turn allows us to express the search time bound in terms of the parameter d . Note that the combinatorial game presented here is different from the known approaches for balls and bins games (see e.g., [3]), since in those approaches the bins are considered static and the distribution of balls uniform. On the contrary, the bins in our game are random variables since the distribution of balls is unknown. This also makes the initialization of the game a non-trivial task which is tackled by firstly sampling a number of balls and then determining appropriate bins which allow the almost uniform distribution of balls into them.

2 Preliminaries

In this paper we consider the unit-cost RAM with a word length of w bits, which models what we program in imperative programming languages such as C. The words of RAM are addressable and these addresses are stored in memory words, imposing that $w \geq \log n$. As a result, the universe U consists of integers (or reals represented as floating point numbers; see [2]) in the range $[0, 2^w - 1]$. It is also assumed that the RAM can perform the standard AC^0 operations of addition, subtraction, comparison, bitwise Boolean operations and shifts, as well as multiplications in constant worst-case time on $O(w)$ -bit operands.

In the following, we make use of another search tree data structure on a RAM called q^* -heap [18]. Let M be the current number of elements in the q^* -heap and let N be an upper bound on the maximum number of elements ever stored in the q^* -heap. Then, insertion, deletion and search operations are carried out in $O(1 + \log M / \log \log N)$ worst-case time after an $O(N)$ preprocessing overhead. Choosing $M = \text{polylog}(N)$, all operations are performed in $O(1)$ time.

In the top level of our data structure we use a tree structure, called *static interpolation search tree*, which is an explicit version of the static implicit tree used in [1] and that uses the REP and ID arrays associated with the nodes of IST. More precisely, the *static interpolation search tree* can be fully characterized by three nondecreasing functions $H(n)$, $R(n)$ and $I(n)$. A static interpolation search tree containing n elements has height $H(n)$, the root has out-degree $R(n)$, and there is an ID array associated with the root that has size $I(n) = n \cdot g(H(n))$ such that $\sum_{i=1}^{\infty} g(i) = \Theta(1)$. To guarantee the height of $H(n)$, it should hold that $n/R(n) = H^{-1}(H(n) - 1)$. The children of the root have $n' = \Theta(n/R(n))$ leaves. Their height will be $H(n') = H(n) - 1$, their out-degree is $R(n') = \Theta(H^{-1}(H(n) - 1)/H^{-1}(H(n) - 2))$, and $I(n') = n' \cdot g(H(n'))$. In general, for an internal node v at depth i containing n_i leaves in the subtree rooted at v , we have that $R(n_i) = \Theta(H^{-1}(H(n) - i + 1)/H^{-1}(H(n) - i))$, and $I(n_i) = n_i \cdot g(H(n) - i)$. As in the case of IST [13], each internal node is associated with an array of sample

elements REP, one for each of its subtrees, and an ID array. By using the ID array, we can interpolate the REP array to determine the subtree in which the search procedure will continue. In particular, the ID array for node v is an array $ID[1..m]$, where m is some integer, with $ID[i] = j$ iff $REP[j] < \alpha + i(\beta - \alpha)/m \leq REP[j + 1]$, where α and β are the minimum and the maximum element, resp., stored in the subtree rooted at v . Let x be the element we seek. To interpolate REP, compute the index $j = ID[\lfloor ((x - \alpha)/(\beta - \alpha)) \rfloor]$, and then scan the REP array from $REP[j + 1]$ until the appropriate subtree is located. For each node we explicitly maintain parent, child, and sibling pointers. Pointers to sibling nodes will be alternatively referred to as *level links*. The required pointer information can be easily incorporated in the construction of the static interpolation search tree. Throughout the paper, we say that an event E occurs *with high probability* (w.h.p.) if $Pr[E] = 1 - o(1)$.

3 The Data Structure

The data structure consists of two separate structures T_1 and T_2 . T_2 is attached a flag *active* denoting whether this structure is valid subject to searches and updates, or invalid. Between two global reconstructions of the data structure, T_1 stores all available elements while T_2 either stores all elements (*active*=TRUE) or a past instance of the set of elements (*active*=FALSE). T_1 is a finger search tree implemented as in [2]. In this way, we can always guarantee worst-case time bounds for searches and updates. In the following we focus on T_2 .

T_2 is a two-level data structure, similar to the Augmented Sampled Forest (ASF) presented in [1], but with the following differences: (a) we use the static interpolation search tree defined in Section 2; (b) we implement the buckets associated with the leaves of the static interpolation search tree using q^* -heaps, instead of simple binary search trees; (c) our search procedure does not start from the root of the tree, but we are guided by a finger f to start from an arbitrary leaf; and (d) our reconstruction procedure to maintain our data structure is quite different from that used in [1]. More specifically, let S_0 be the set of elements to be stored where the elements take values in $[a, b]$. The two levels of T_2 are as follows. The bottom level is a set of ρ buckets. Each bucket $\mathcal{B}_i$, $1 \leq i \leq \rho$, stores a subset of elements and is represented by the element $rep(i) = \max\{x : x \in \mathcal{B}_i\}$. The set of elements stored in the buckets constitute an ordered collection $\mathcal{B}_1, \dots, \mathcal{B}_\rho$ such that $\max\{x : x \in \mathcal{B}_i\} < \min\{y : y \in \mathcal{B}_{i+1}\}$ for all $1 \leq i \leq \rho - 1$. In other words, $\mathcal{B}_i = \{x : x \in (rep(i - 1), rep(i)]\}$, for $2 \leq i \leq \rho$, and $\mathcal{B}_1 = \{x : x \in [rep(0), rep(1)]\}$, where $rep(0) = a$ and $rep(\rho) = b$. Each $\mathcal{B}_i$ is implemented as a q^* -heap [18]. The top level data structure is a static interpolation search tree that stores all elements.

Our data structure is maintained by incrementally performing global reconstructions [14]. More precisely, let S_0 be the set of stored elements at the latest reconstruction, and assume that $S_0 = \{x_1, \dots, x_{n_0}\}$ in sorted order. The reconstruction is performed as follows. We partition S_0 into two sets S_1 and S_2 , where $S_1 = \{x_{i \cdot \ln n_0} : i = 1, \dots, \frac{n_0}{\ln n_0} - 1\} \cup \{b\}$, and $S_2 = S_0 - S_1$. The i -th element

of S_1 is the representative $rep(i)$ of the i -th bucket $\mathcal{B}_i$, where $1 \leq i \leq \rho$ and $\rho = |S_1| = \frac{n_0}{\ln n_0}$. An element $x \in S_2$ is stored twice: (i) In the appropriate bucket $\mathcal{B}_i$, iff $rep(i-1) < x \leq rep(i)$, for $2 \leq i \leq \frac{n_0}{\ln n_0}$; otherwise ($x \leq rep(1)$), x is stored in $\mathcal{B}_1$. (ii) As a leaf in the top level structure where it is marked redundant and is equipped with a pointer to the representative of the bucket to which it belongs. We also mark as redundant all internal nodes of the top level structure that span redundant leaves belonging to the same bucket and equip them with a pointer to the representative of the bucket. The reason we store the elements of S_2 twice is to ensure that all elements are drawn from the same μ -random distribution and hence we can safely apply the analysis presented in [1,13]. Also, the reason for this kind of representatives will be explained in Section 5. Note that, after reconstruction, each new element is stored only in the appropriate bucket. Each time the number of updates exceeds rn_0 , where r is an arbitrary constant, the whole data structure is reconstructed. Let n be the number of stored elements at this time. After the reconstruction, the number of buckets is equal to $\frac{n}{\ln n}$ and the value of the parameter N , used for the implementation of $\mathcal{B}_i$ with a q^* -heap, is n . Immediately after the reconstruction, if every bucket stores less than $\text{polylog}(n)$ elements, then $active=TRUE$, otherwise $active=FALSE$.

In order to insert/delete an element immediately to the right of an existing element f , we insert/delete the element to/from T_1 (using the procedures in [2]), and we insert/delete the element to/from the appropriate bucket of T_2 if $active=TRUE$ (using the procedures in [18]). If during an insertion in a bucket of T_2 , the number of stored elements becomes greater than $\text{polylog}(n)$, then $active=FALSE$. The search procedure for locating an element x in the data structure, provided that a finger f to some element is given, is carried out as follows. If $active=TRUE$, then we search in parallel both structures and we stop when we first locate the element, otherwise we only search in T_1 . The search procedure in T_1 is carried out as in [2]. The search procedure in T_2 involves a check as to whether x is to the left or to the right of f . Assume, without loss of generality, that x is to the right of f . Then, we have two cases: (1) Both elements belong to the same bucket $\mathcal{B}_i$. In this case, we just retrieve from the q^* -heap that implements $\mathcal{B}_i$ the element with key x . (2) The elements are stored in different buckets $\mathcal{B}_i$ and $\mathcal{B}_j$ containing f and x respectively. In this case, we start from $rep(i)$ and we walk towards the root of the static interpolation search tree. Assuming that we reach a node v , we check whether x is stored in a descendant of v or in the right neighbour z of v . This can be easily accomplished by checking the boundaries of the REP arrays of both nodes. If they are not stored in the subtrees of v and z , then we proceed to the parent of v , otherwise we continue the search in the particular subtree using the ID and REP arrays. When a redundant node is reached, we follow its associated pointer to the appropriated bucket.

4 Analysis of Time and Space Complexity

In this section we analyze the time complexities of the search and update operations. We start with the case of $(n/(\log \log n)^{1+\epsilon}, n^{1-\delta})$ -smooth densities, and

later on discuss how our result can be extended to the general case. The tree structure T_2 is updated and queried only in the case where all of its buckets have size $\text{polylog}(n)$ ($\text{active}=\text{TRUE}$), where n is the number of elements in the latest reconstruction. By this and by using some arguments of the analysis in [2] and [18] the following lemma is immediate.

Lemma 1. *The preprocessing time and the space usage of our data structure is $\Theta(n)$. The update operations are performed in $O(1)$ worst-case time.*

The next theorem gives the time complexity of our search operation.

Theorem 1. *Suppose that the top level of T_2 is a static interpolation search tree with parameters $R(s_0) = (s_0)^{1-\delta}$, $I(s_0) = s_0/(\log \log s_0)^{1+\epsilon}$, where $\epsilon > 0$, $0 < \delta < 1$, and $s_0 = \frac{n_0}{\ln n_0}$ with $\text{active}=\text{TRUE}$. Then, the time complexity of a search operation is equal to $O(\min\{\frac{\log |\mathcal{B}_i|}{\log \log n} + \frac{\log |\mathcal{B}_j|}{\log \log n} + \log \log d, \sqrt{\log d / \log \log d}\})$, where $\mathcal{B}_i$ and $\mathcal{B}_j$ are the buckets containing the finger f and the search element x respectively, d denotes the number of buckets between $\mathcal{B}_i$ and $\mathcal{B}_j$, and n denotes the current number of elements.*

Proof (Sketch). Since $\text{active}=\text{TRUE}$, the search time is the minimum of searching in each of T_1 and T_2 . Searching the former equals $O(\sqrt{\log d / \log \log d})$. It is not hard to see that the search operation in T_2 involves at most two searches in buckets $\mathcal{B}_i$ and $\mathcal{B}_j$, and the traversal of internal nodes of the static interpolation search tree, using ancestor pointers, level links and interpolation search. This traversal involves ascending and descending a subtree of at most d leaves and height $O(\log \log d)$, and we can prove (by modifying the analysis in [1,13]) that the time spent at each node during descend is $O(1)$ w.h.p. $\square$

To prove that the data structure has a low expected search time with high probability we introduce a combinatorial game of balls and bins with deletions (Section 5). To get the desirable time complexities w.h.p., we provide upper and lower bounds on the number of elements in a bucket and we show that no bucket gets empty (see Theorem 6). Combining Theorems 1 and 6 we get the main result of the paper.

Theorem 2. *There exists a finger search tree with $O(\log \log d)$ expected search time with high probability for μ -random insertions and random deletions, where μ is a $(n/(\log \log n)^{1+\epsilon}, n^{1-\delta})$ -smooth density for $\epsilon > 0$ and $0 < \delta < 1$, and d is the distance between the finger and the search element. The space usage of the data structure is $\Theta(n)$, the worst-case update time is $O(1)$, and the worst-case search time is $O(\sqrt{\log d / \log \log d})$.*

We can generalize our results to hold for the class of $(n \cdot g(H(n)), H^{-1}(H(n) - 1))$ -smooth densities considered in [1], where $H(n)$ is an increasing function representing the height of the static interpolation tree and g is a function satisfying $\sum_{i=1}^{\infty} g(i) = \Theta(1)$, thus being able to achieve $o(\log \log d)$ expected time complexity, w.h.p., for several distributions. The generalization follows the proof of Theorem 1 by showing that the subtree of the static IST has now height $O(H(d))$, implying the same traversal time w.h.p. (details in the full paper [9]).

Theorem 3. *There exists a finger search tree with $\Theta(H(d))$ expected search time with high probability for μ -random insertions and random deletions, where d is the distance between the finger and the search element, and μ is a $(n \cdot g(H(n)), H^{-1}(H(n) - 1))$ -smooth density, where $\sum_{i=1}^{\infty} g(i) = \Theta(1)$. The space usage of the data structure is $\Theta(n)$, the worst-case update time is $O(1)$, and the worst-case search time is $O(\sqrt{\log d / \log \log d})$.*

For example, the density $\mu[0, 1](x) = -\ln x$ is $(n/(\log^* n)^{1+\epsilon}, \log^2 n)$ -smooth, and for this density $R(n) = n/\log^2 n$. This means that the height of the tree with n elements is $H(n) = \Theta(\log^* n)$ and the method of [1] gives an expected search time complexity of $\Theta(\log^* n)$. However, by applying Theorem 3, we can reduce the expected time complexity for the search operation to $\Theta(\log^* d)$ and this holds w.h.p. If μ is bounded, then it is $(n, 1)$ -smooth and hence $H(n) = O(1)$, implying the same expected search time with [1] but w.h.p.

5 A Combinatorial Game of Bins and Balls with Deletions

In this section we describe a balls-in-bins random process that models each update operation in the structure T_2 presented in Section 3. Consider the structure T_2 immediately after the latest reconstruction. It contains the set S_0 of n elements (we shall use n for notational simplicity) which are drawn randomly according to the distribution $\mu(\cdot)$ from the interval $[a, b]$. The next reconstruction is performed after rn update operations on T_2 , where r is a constant. Each update operation is either a uniformly at random deletion of an existing element from T_2 , or a μ -random insertion of a new element from $[a, b]$ into T_2 . To model the update operations as a balls-in-bins random process, we do the following.

We represent each selected element from $[a, b]$ as a *ball*. We partition the interval $[a, b]$ into $\rho = \frac{n}{\ln n}$ parts $[rep(0), rep(1)] \cup (rep(1), rep(2)] \cup \dots \cup (rep(\rho - 1), rep(\rho)]$, where $rep(0) = a$, $rep(\rho) = b$, and $\forall i = 1, \dots, \rho - 1$, the elements $rep(i) \in [a, b]$ are those defined in Section 3. We represent each of these ρ parts as a distinct *bin*. During each of the rn insertion/deletion operations in T_2 , a μ -random ball $x \in [a, b]$ is inserted in (deleted from) the i -th bin $\mathcal{B}_i$ iff $rep(i - 1) < x \leq rep(i)$, $i = 2, \dots, \rho$; otherwise x , is inserted in (deleted from) $\mathcal{B}_1$.

Our aim is to prove that w.h.p. the maximum load of any bin is $O(\ln n)$, and that no bin remains empty as $n \rightarrow \infty$. If we were knowing the distribution $\mu(\cdot)$, then we could partition the interval $[a, b]$ into ρ distinct bins $[rep_\mu(0), rep_\mu(1)] \cup (rep_\mu(1), rep_\mu(2)] \cup \dots \cup (rep_\mu(\rho - 1), rep_\mu(\rho)]$, with $rep_\mu(0) = a$ and $rep_\mu(\rho) = b$, such that a μ -random ball x would be equally likely to belong into any of the ρ corresponding bins with probability $Pr[x \in (rep_\mu(i - 1), rep_\mu(i)]] = \int_{rep_\mu(i-1)}^{rep_\mu(i)} \mu(t) dt = \frac{1}{\rho} = \frac{\ln n}{n}$. The above expression implies that the sequence $rep_\mu(0), \dots, rep_\mu(\rho)$ makes the event “insert (delete) a μ -random (random) element x into (from) the structure” equivalent to the event “throw (delete) a ball uniformly at random into (from) one of ρ distinct bins”. Such a uniform distri-

bution of balls into bins is well understood and it is folklore to find conditions such that no bin remains empty and no bin gets more than $O(\ln n)$ balls.

Unfortunately, the probability density $\mu(\cdot)$ is unknown. Consequently, our goal is to *approximate* the unknown sequence $rep_\mu(0), \dots, rep_\mu(\rho)$ with a sequence $rep(0), \dots, rep(\rho)$, that is, to partition the interval $[a, b]$ into ρ parts $[rep(0), rep(1)] \cup (rep(1), rep(2)] \cup \dots \cup (rep(\rho - 1), rep(\rho)]$, aiming to prove that each bin (part) will have the key property: $Pr[x \in (rep(i - 1), rep(i)]] = \int_{rep(i-1)}^{rep(i)} \mu(t)dt = \Theta\left(\frac{1}{\rho}\right) = \Theta\left(\frac{\ln n}{n}\right)$. The sequence $rep(0), \dots, rep(\rho)$ makes the event “insert (delete) a μ -random (random) element x into (from) the structure” equivalent to the event “throw (delete) a ball *almost* uniformly at random into one of ρ distinct bins”. This fact will become the cornerstone in our subsequent proof that no bin remains empty and almost no bin gets more than $\Theta(\ln n)$ balls.

The basic insight of our approach is illustrated by the following random game. Consider the part of the horizontal axis spanned by $[a, b]$, which will be referred to as the $[a, b]$ *axis*. Suppose that only a wise man knows the positions on the $[a, b]$ axis of the sequence $rep_\mu(0), \dots, rep_\mu(\rho)$, referred to as the *red dots*. Next, perform n independent insertions of μ -random elements from $[a, b]$ (this is the role of the set S_0). In each insertion of an element x , we add a *blue dot* in its position on the $[a, b]$ axis. At the end of this random game we have a total of n blue dots in this axis. Now, the wise man reveals the red dots on the $[a, b]$ axis, i.e., the sequence $rep_\mu(0), \dots, rep_\mu(\rho)$. If we start counting the blue dots *between* any two consecutive red dots $rep_\mu(i - 1)$ and $rep_\mu(i)$, we almost always find that there are $\ln n + o(1)$ blue dots. This is because the number X_i^μ of μ -random elements (blue dots) selected from $[a, b]$ that belong in $(rep_\mu(i - 1), rep_\mu(i)]$, $i = 1, \dots, \rho$, is a Binomial random variable, $X_i^\mu \sim B(n, \frac{1}{\rho} = \frac{\ln n}{n})$, which is sharply concentrated to its expectation $E[X_i^\mu] = \ln n$.

The above discussion suggests the following procedure for constructing the sequence $rep(0), \dots, rep(\rho)$. Partition the sequence of n blue dots on the $[a, b]$ axis into $\rho = \frac{n}{\ln n}$ parts, each of size $\ln n$. Set $rep(0) = a$, $rep(\rho) = b$, and set as $rep(i)$ the $i \cdot \ln n$ -th blue dot, $i = 1, \dots, \rho - 1$. Call this procedure **Red-Dots**.

The above intuitive argument does not imply that $\lim_{n \rightarrow \infty} rep(i) = rep_\mu(i)$, $\forall i = 0, \dots, \rho$. Clearly, since $rep_\mu(i)$, $i = 0, \dots, \rho$, is a real number, the probability that at least one blue dot *hits* an invisible red dot is insignificant. The above argument stresses on the following fact whose proof can be found in [9].

Theorem 4. *Let $rep(0), rep(1), \dots, rep(\rho)$ be the output of procedure Red-Dots, and let $p_i(n) = \int_{rep(i-1)}^{rep(i)} \mu(t)dt$. Then:*

$$Pr \left[\exists i \in \{1, \dots, m\} : p_i(n) \neq \Theta\left(\frac{1}{\rho}\right) = \Theta\left(\frac{\ln n}{n}\right) \right] \rightarrow 0.$$

The above discussion and Theorem 4 imply the following.

Corollary 1. *If n elements are μ -randomly selected from $[a, b]$, and the sequence $rep(0), \dots, rep(\rho)$ from those elements is produced by procedure Red-Dots, then this sequence partitions the interval $[a, b]$ into ρ distinct bins (parts) $[rep(0), rep(1)] \cup (rep(1), rep(2)] \cup \dots \cup (rep(\rho - 1), rep(\rho)]$ such that a ball $x \in [a, b]$*

can be thrown (deleted) independently of any other ball in $[a, b]$ into (from) any of the bins with probability $p_i(n) = \Pr[x \in (\text{rep}(i-1), \text{rep}(i))] = \frac{c_i \ln n}{n}$, where $i = 1, \dots, \rho$ and c_i is a positive constant.

Definition 1. Let $c = \min_i \{c_i\}$ and $C = \max_i \{c_i\}$, $i = 1, \dots, \rho$, where $c_i = \frac{np_i(n)}{\ln n}$.

We now turn to the randomness properties in each of the rn subsequent insertion/deletion operations on the structure T_2 (r is a constant).

Observe that before the process of rn insertions/deletions starts, each bin $\mathcal{B}_i$ (i.e., part $(\text{rep}(i-1), \text{rep}(i))$) contains exactly $\ln n$ balls (blue dots on the $[a, b]$ axis) of the n initial balls of the set S_0 . For convenience, we analyze a slightly different process of the subsequent rn insertions/deletions. Delete all elements (balls) of S_0 except for the representatives $\text{rep}(0), \text{rep}(1), \dots, \text{rep}(\rho)$ of the ρ bins. Then, insert μ -randomly n/c (see Definition 1) new elements (balls) and subsequently start performing the rn insertions/deletions. Since the n/c new balls are thrown μ -randomly into the ρ bins $[\text{rep}(0), \text{rep}(1)] \cup (\text{rep}(1), \text{rep}(2)] \cup \dots \cup (\text{rep}(\rho-1), \text{rep}(\rho)]$, by Corollary 1 the initial number of balls into $\mathcal{B}_i$ is a Binomial random variable that obeys $B(n/c, p_i(n))$, $i = 1, \dots, \rho$, instead of being fixed to the value $\ln n$. Clearly, if we prove that for this process no bin remains empty and does not contain more than $O(\ln n)$ balls, then this also holds for the initial process.

Let the random variable $M(j)$ denote the number of balls existing in structure T_2 at the end of the j -th insertion/deletion operation, $j = 0, \dots, rn$. Initially, $M(0) = n/c$. The next useful lemma allows us to keep track of the statistics of an arbitrary bin. Part (i) follows by Corollary 1 and an induction argument, while part (ii) is an immediate consequence of part (i).

Lemma 2. (i) Suppose that at the end of j -th insertion/deletion operation there exist $M(j)$ distinct balls that are μ -randomly distributed into the ρ distinct bins. Then, after the $(j+1)$ -th insertion/deletion operation the $M(j+1)$ distinct balls are also μ -randomly distributed into the ρ distinct bins. (ii) Let the random variable $Y_i(j)$ with $(i, j) \in \{1, \dots, \rho\} \times \{0, \dots, rn\}$ denote the number of balls that the i -th bin contains at the end of the j -th operation. Then, $Y_i(j) \sim B(M(j), p_i(n))$.

To study the dynamics of $M(j)$ at the end of j -th operation, observe that in each operation, a ball is either inserted with probability $p > 1/2$, or is deleted with probability $1 - p$. $M(j)$ is a discrete random variable which has the nice property of sharp concentration to its expected value, i.e., it has small deviation from its mean compared to the total number of operations. In the following, instead of working with the actual values of j and $M(j)$, we shall use their scaled (divided by n) values t and $m(t)$, resp., that is, $t = \frac{j}{n}$, $m(t) = \frac{M(tn)}{n}$, with range $(t, m(t)) \in [0, r] \times [1, m(r)]$. The sharp concentration property of $M(j)$ leads to the following theorem (whose proof can be found in [9]).

Theorem 5. For each operation $0 \leq t \leq r$, the scaled number of balls that are distributed into the $\frac{n}{\ln(n)}$ bins at the end of the t -th operation equals $m(t) = (2p - 1)t + o(1)$, w.h.p.

Remark 1. Observe that for $p > 1/2$, $m(t)$ is an increasing positive function of the scaled number t of operations, that is, $\forall t \geq 0$, $M(tn) = m(t)n \geq M(0) = m(0)n = n/c$. This implies that if no bin remains empty before the process of rn operations starts, since for $p > 1/2$ the balls accumulate as the process evolve, then no bin will remain empty in each subsequent operation. This is important on proving part (i) of Theorem 6.

Finally, we turn to the statistics of the bins. We prove that before the first operation, and for all subsequent operations, w.h.p., no bin remains empty. Furthermore, we prove that during each step the maximum load of any bin is $\Theta(\ln(n))$ w.h.p. For the analysis below we make use of the Lambert function $LW(x)$, which is the analytic at zero solution with respect to y of the equation: $ye^y = x$ (see [4]). Recall also that during each operation $j = 0, \dots, rn$ with probability $p > 1/2$ we insert a μ -random ball $x \in [a, b]$, and with probability $1 - p$ we delete an existing ball from the current $M(j)$ balls that are stored in the structure T_2 .

Theorem 6. (i) For each operation $0 \leq t \leq r$, let the random variable $X(t)$ denote the current number of empty bins. If $p > 1/2$, then for each operation t , $E[X(t)] \rightarrow 0$. (ii) At the end of operation t , let the random variable $Z_\kappa(t)$ denote the number of bins with load at least $\kappa \ln(n)$, where $\kappa = \kappa(t)$ satisfies $\kappa \geq (-Cm(t) + 2)/(C \cdot LW(-\frac{Cm(t)-2}{Cm(t)e})) = O(1)$, and C is the positive constant defined in Definition 1. If $p > 1/2$, then for each operation t , $E[Z_\kappa(t)] \rightarrow 0$.

Proof. (i) Recall the definitions of the positive constants c and C (Definition 1). From Lemma 2, $\forall i = 1, \dots, \rho = \frac{n}{\ln(n)}$, it holds:

$$Pr[Y_i(t) = 0] \leq \left(1 - c \frac{\ln(n)}{n}\right)^{m(t)n} \sim e^{-cm(t) \ln(n)} = \frac{1}{n^{cm(t)}}. \quad (1)$$

From Eq. (1), by linearity of expectation, we obtain:

$$E[X(t) \mid m(t)] \leq \sum_{i=1}^{\rho} Pr[Y_i(t) = 0] \leq \frac{n}{\ln(n)} \cdot \frac{1}{n^{cm(t)}}. \quad (2)$$

From Theorem 5 and Remark 1 it holds: $\forall t \geq 0$, $\frac{1}{n^{cm(t)}} \leq \frac{1}{n^{cm(0)}} = \frac{1}{n}$. This inequality implies that in order to show for each operation t that the expected number $E[X(t) \mid m(t)]$ of empty bins vanishes, it suffices to show that before the process starts, the expected number $E[X(0) \mid m(0)]$ of empty bins vanishes. In this line of thought, from Theorem 5, Eq. (2) becomes,

$$E[X(0) \mid m(0)] \leq \frac{n}{\ln(n)} \cdot \frac{1}{n^{cm(0)}} = \frac{n}{\ln(n)} \cdot \frac{1}{n} = \frac{1}{\ln(n)} \rightarrow 0.$$

Finally, from Markov's inequality, we obtain

$$Pr[X(t) > 0 \mid m(t)] \leq E[X(t) \mid m(t)] \leq E[X(0) \mid m(0)] \rightarrow 0.$$

(ii) In the full paper [9] due to space limitations. □

References

1. A. Andersson and C. Mattson. Dynamic Interpolation Search in $o(\log \log n)$ Time. In *Proc. ICALP'93*.
2. A. Anderson and M. Thorup. Tight(er) Worst-case Bounds on Dynamic Searching and Priority Queues. In *Proc. 32nd ACM Symposium on Theory of Computing* – STOC 2001, pp. 335–342. ACM, 2000.
3. R. Cole, A. Frieze, B. Maggs, M. Mitzenmacher, A. Richa, R. Sitaraman, and E. Upfal. On Balls and Bins with Deletions. In *Randomization and Approximation Techniques in Computer Science* – RANDOM'98, Lecture Notes in Computer Science Vol. 1518 (Springer-Verlag, 1998), pp. 145–158.
4. R.M. Corless, G.H. Gonnet, D.E.G. Hare, D.J. Jeffrey, and D.E. Knuth. On the Lambert W Function. *Advances in Computational Mathematics* 5:329–359, 1996.
5. P. Dietz and R. Raman. A Constant Update Time Finger Search Tree. *Information Processing Letters*, 52:147–154, 1994.
6. G. Frederickson. Implicit Data Structures for the Dictionary Problem. *Journal of the ACM* 30(1):80–94, 1983.
7. G. Gonnet, L. Rogers, and J. George. An Algorithmic and Complexity Analysis of Interpolation Search. *Acta Informatica* 13(1):39–52, 1980.
8. A. Itai, A. Konheim, and M. Rodeh. A Sparse Table Implementation of Priority Queues. In *Proc. ICALP'81*, Lecture Notes in Computer Science Vol. 115 (Springer-Verlag 1981), pp. 417–431.
9. A. Kaporis, C. Makris, S. Sioutas, A. Tsakalidis, K. Tsichlas, and C. Zaroliagis. Improved Bounds for Finger Search on a RAM. Tech. Report TR-2003/07/01, Computer Technology Institute, Patras, July 2003.
10. D.E. Knuth. Deletions that preserve randomness. *IEEE Trans. Softw. Eng.* 3:351–359, 1977.
11. C. Levcopoulos and M.H. Overmars. A Balanced Search Tree with $O(1)$ Worst Case Update Time. *Acta Informatica*, 26:269–277, 1988.
12. K. Mehlhorn and A. Tsakalidis. *Handbook of Theoretical Computer Science – Vol I: Algorithms and Complexity*, Chapter 6: Data Structures, pp. 303–341, The MIT Press, 1990.
13. K. Mehlhorn and A. Tsakalidis. Dynamic Interpolation Search. *Journal of the ACM*, 40(3):621–634, July 1993.
14. M. Overmars, J. Leeuwen. Worst Case Optimal Insertion and Deletion Methods for Decomposable Searching Problems. *Information Processing Letters*, 12(4):168–173.
15. Y. Pearl, A. Itai, and H. Avni. Interpolation Search – A $\log \log N$ Search. *Communications of the ACM* 21(7):550–554, 1978.
16. W.W. Peterson. Addressing for Random Storage. *IBM Journal of Research and Development* 1(4):130–146, 1957.
17. D.E. Willard. Searching Unindexed and Nonuniformly Generated Files in $\log \log N$ Time. *SIAM Journal of Computing* 14(4):1013–1029, 1985.
18. D.E. Willard. Applications of the Fusion Tree Method to Computational Geometry and Searching. In *Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms* – SODA'92, pp. 286–295, 1992.
19. A.C. Yao and F.F. Yao. The Complexity of Searching an Ordered Random Table. In *Proc. 17th IEEE Symp. on Foundations of Computer Science* – FOCS'76, pp. 173–177, 1976.

The Voronoi Diagram of Planar Convex Objects^{*}

Menelaos I. Karavelas¹ and Mariette Yvinec²

¹ University of Notre Dame, Computer Science and Engineering Department,
Notre Dame, IN 46556, U.S.A.

`mkaravel@cse.nd.edu`

² INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93,
06902 Sophia-Antipolis Cedex, France

`Mariette.Yvinec@sophia.inria.fr`

Abstract. This paper presents a dynamic algorithm for the construction of the Euclidean Voronoi diagram of a set of convex objects in the plane. We consider first the Voronoi diagram of smooth convex objects forming pseudo-circles set. A pseudo-circles set is a set of bounded objects such that the boundaries of any two objects intersect at most twice. Our algorithm is a randomized dynamic algorithm. It does not use a conflict graph or any sophisticated data structure to perform conflict detection. This feature allows us to handle deletions in a relatively easy way. In the case where objects do not intersect, the randomized complexity of an insertion or deletion can be shown to be respectively $O(\log^2 n)$ and $O(\log^3 n)$. Our algorithm can easily be adapted to the case of pseudo-circles sets formed by piecewise smooth convex objects. Finally, given any set of convex objects in the plane, we show how to compute the restriction of the Voronoi diagram in the complement of the objects' union.

1 Introduction

Given a set of sites and a distance function from a point to a site, a Voronoi diagram can be roughly described as the partition of the space into cells that are the locus of points closer to a given site than to any other site. Voronoi diagrams have proven to be useful structures in various fields such as astronomy, crystallography, biology etc. Voronoi diagrams have been extensively studied. See for example the survey by Aurenhammer and Klein [1] or the book by Okabe, Boots, Sugihara and Chiu [2]. The early studies were mainly concerned with point sites and the Euclidean distance. Subsequent studies considered extended sites such as segments, lines, convex polytopes and various distances such as L_1 or L_∞ or any distance defined by a convex polytope as unit ball. While the complexity and the related algorithmic issues of Voronoi diagrams for extended sites in higher dimensions is still not completely understood, as witnessed in the

^{*} Work partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

recent works by Koltun and Sharir [3,4], the planar cases are now rather well mastered, at least for linear objects.

The rising need for handling curved objects triggered further works for the planar cases. Klein et al. [5,6] set up a general framework of *abstract Voronoi diagrams* which covers a large class of planar Voronoi diagrams. They provided a randomized incremental algorithm to construct diagrams of this class. Alt and Schwarzkopf [7] handled the case of generic planar curves and described a randomized algorithm for this case. Since they handle curves, they cannot handle objects with non-empty interior, which is our focus. Their algorithm is incremental but does not work in-line (it requires the construction of a Delaunay triangulation with one point on each curve before the curve segments are really treated). Another closely related work is that by McAllister, Krikpatrick and Snoeyink [8], which deals with the Voronoi diagrams of disjoint convex polygons. The algorithm presented treats the convex polygons as objects, rather than as collections of segments; it follows the sweep-line paradigm, thus it is not dynamic. Moreover, the case of intersecting convex polygons is not considered. The present paper deals with the Euclidean Voronoi diagram of planar smooth or piecewise smooth convex objects, and generalizes a previous work of the same authors on the Voronoi diagram of circles [9].

Let p be a point and A be a bounded convex object in the Euclidean plane $\mathbb{E}^2$. We define the distance $\delta(p, A)$ from p to A to be:

$$\delta(p, A) = \begin{cases} \min_{x \in \partial A} \|p - x\|, & p \notin A \\ -\min_{x \in \partial A} \|p - x\|, & p \in A \end{cases}$$

where ∂A denotes the boundary of A and $\|\cdot\|$ denotes the Euclidean norm. Given the distance $\delta(\cdot, \cdot)$ and a set of convex objects $\mathcal{A} = \{A_1, \dots, A_n\}$, the *Voronoi diagram* $\mathcal{V}(\mathcal{A})$ is the planar partition into cells, edges and vertices defined as follows. The Voronoi cell of an object A_i is the set of points which are closer to A_i than to any other object in $\mathcal{A}$. Voronoi edges are maximal connected sets of points equidistant to two objects in $\mathcal{A}$ and closer to these objects than to any other in $\mathcal{A}$. Voronoi vertices are points equidistant to at least three objects of $\mathcal{A}$ and closer to these objects than to any other object in $\mathcal{A}$.

We first consider Voronoi diagrams for special collections of smooth convex objects called *pseudo-circles sets*. A pseudo-circles set is a set of bounded objects such that the boundaries of any two objects in the set have at most two intersection points. In the sequel, unless specified otherwise, we consider pseudo-circles sets formed by smooth convex objects, and we call them *smooth convex pseudo-circles sets*, or *sc-pseudo-circles sets* for short.

Let A be a convex object. A line L is a *supporting line* of A iff A is included in one of the closed half-planes bounded by L , and $\partial A \cap L$ is not empty. Given two convex objects A_i and A_j , a line L is a (*common*) *supporting line* of A_i and A_j iff L is a supporting line of A_i and A_j , such that A_i and A_j are both included in the same half-plane bounded by L . In this paper, we first deal with smooth bounded convex objects forming pseudo-circles sets. Any two objects in such a set have at most two common supporting lines. Two convex objects

have no common supporting line if one is included in the other. They have two common supporting lines if they are either disjoint or properly intersecting at two points (a proper intersection point is a point where the boundaries are not only touching but also crossing each other) or externally tangent (which means that their interiors are disjoint and their boundaries share a common tangent point). Two objects forming a pseudo-circles set may also be internally tangent, meaning that one is included in the other and their boundaries share one or two common points. Then they have, respectively, one or two common supporting lines. A pseudo-circles set is said to be in *general position* if there is no pair of tangent objects. In fact, tangent objects which are properly intersecting at their common tangent point or externally tangent objects do not harm our algorithm and we shall say that a pseudo-circles set is in general position when there is no pair of internally tangent objects.

The algorithm that we propose for the construction of the Voronoi diagram of sc-pseudo-circles sets in general position is a dynamic one. It is a variant of the incremental randomized algorithm proposed by Klein et al. [6]. The data structures used are simple, which allows us to perform not only insertions but also deletions of sites in a relatively easy way. When input sites are allowed to intersect each other, it is possible for a site to have an empty Voronoi cell. Such a site is called *hidden*, otherwise *visible*. Our algorithm handles hidden sites. The detection of the first conflict or the detection of a hidden site is performed through closest site queries. Such a query can be done by either a simple walk in the Voronoi diagram or using a hierarchy of Voronoi diagrams, i.e., a data structure inspired from the Delaunay hierarchy of Devillers [10].

To analyze the complexity of the algorithm, we assume that each object has constant complexity, which implies that each operation involving a constant number of objects is performed in constant time. We show that if sites do not intersect, the randomized complexity of updating a Voronoi diagram with n sites is $O(\log^2 n)$ for an insertion and $O(\log^3 n)$ for a deletion. The complexities of insertions and deletions are more involved when sites intersect.

We then extend our results by firstly dropping the hypothesis of general position and secondly by dealing with pseudo-circles sets formed by convex objects whose boundaries are only piecewise smooth. Using this extension, we can then build the Voronoi diagram of any set $\mathcal{A}$ of convex objects in the complement of the objects' union (i.e., in free space). This is done by constructing a new set of objects $\mathcal{A}'$, which is a pseudo-circles set of piecewise smooth convex objects and such that the Voronoi diagrams $\mathcal{V}(\mathcal{A})$ and $\mathcal{V}(\mathcal{A}')$ coincide in free space.

The rest of the paper is structured as follows. In Section 2 we study the properties of the Voronoi diagram of sc-pseudo-circles sets in general position, and show that such a diagram belongs to the class of abstract Voronoi diagrams. In Section 3 we present our dynamic algorithm. Section 4 describes closest site queries, whereas Section 5 deals with the complexity analysis of insertions and deletions. Finally, in Section 6 we discuss the extensions of our approach.

2 The Voronoi Diagram of sc-Pseudo-Circles Sets

In this section we present the main properties of the Voronoi diagram of sc-pseudo-circles sets in general position. We first provide a few definitions and notations. Henceforth, we consider any bounded convex object A_i as closed and we note ∂A_i and A_i° , respectively, the boundary and the interior of A_i .

Let $\mathcal{A} = \{A_1, \dots, A_n\}$ be an sc-pseudo-circles set. The Voronoi cell of an object A is denoted as $V(A)$ and is considered a closed set. The interior and boundary of $V(A)$ are denoted by $V^\circ(A)$ and $\partial V(A)$, respectively. We are going to consider maximal disks either included in a given object A_i or disjoint from A_i° , where the term maximal refers to the inclusion relation. For any point x , we denote by $C_i(x)$ the closed disk centered at x with radius $|\delta(x, A_i)|$. If $x \notin A_i$, $C_i(x)$ is the maximal disk centered at x and disjoint from A_i° . If $x \in A_i$, $C_i(x)$ is the maximal disk centered at x and included in A_i . In the latter case there is a unique maximal disk inside A_i containing $C_i(x)$, which we denote by $M_i(x)$. Finally, the *medial axis* $S(A_i)$ of a bounded convex object A_i is defined as the locus of points that are centers of maximal disks included in A_i .

Let A_i and A_j be two smooth bounded convex objects. The set of points $p \in \mathbb{E}^2$ that are at equal distance from A_i and A_j is called the bisector π_{ij} of A_i and A_j . Theorem 1 ensures that π_{ij} is an one-dimensional set if the two objects A_i and A_j form an sc-pseudo-circles set in general position and justifies the definition of Voronoi edges given above. Theorem 2 ensures that each cell in the Euclidean Voronoi diagram of an sc-pseudo-circles set in general position is simply connected. The proofs of Theorems 1 and 2 below are omitted for lack of space.

Theorem 1 *Let $\{A_i, A_j\}$ be an sc-pseudo-circles set in general position and let π_{ij} be the bisector of A_i and A_j with respect to the Euclidean distance $\delta(\cdot, \cdot)$. Then: (1) if A_i and A_j have no common supporting line, $\pi_{ij} = \emptyset$; (2) if A_i and A_j have two common supporting lines, π_{ij} is a single curve homeomorphic to the open interval $(0, 1)$.*

Theorem 2 *Let $\mathcal{A} = \{A_1, \dots, A_n\}$ be an sc-pseudo-circles set in general position. For each object A_i , we denote by $N(A_i)$ the locus of the centers of maximal disks included in A_i that are not included in the interior of any object in $\mathcal{A} \setminus \{A_i\}$, and by $N^\circ(A_i)$ the locus of the centers of maximal disks included in A_i that are not included in any object in $\mathcal{A} \setminus \{A_i\}$. Then: (1) $N(A_i) = S(A_i) \cap V(A_i)$ and $N^\circ(A_i) = S(A_i) \cap V^\circ(A_i)$; (2) $N(A_i)$ and $N^\circ(A_i)$ are simply connected sets; (3) the Voronoi cell $V(A_i)$ is weakly star-shaped with respect to $N(A_i)$, which means that any point of $V(A_i)$ can be connected to a point in $N(A_i)$ by a segment included in $V(A_i)$. Analogously, $V^\circ(A_i)$ is weakly star-shaped with respect to $N^\circ(A_i)$; (4) $V(A_i) = \emptyset$ iff $N(A_i) = \emptyset$ and $V^\circ(A_i) = \emptyset$ iff $N^\circ(A_i) = \emptyset$.*

In the sequel we say that an object A is *hidden* if $N^\circ(A) = \emptyset$.

In the framework of abstract Voronoi diagrams introduced by Klein [5], the diagram is defined by a set of bisecting curves $B_{i,j}$. In this framework, a set

of bisectors is said to be *admissible* if: (1) each bisector is homeomorphic to a line; (2) the closures of the Voronoi regions covers the entire plane; (3) regions are path connected. (4) two bisectors intersect in at most a finite number of connected components. Let us show that Euclidean Voronoi diagrams of sc-pseudo-circles, such that any pair of objects has exactly two supporting lines, fit into the framework of abstract Voronoi diagrams. Theorems 1 and 2 ensure, respectively, that Conditions 1 and 3 are fulfilled. Condition 2 is granted for any diagram induced by a distance. Condition 4 is a technical condition that we have not explicitly proved. In our case this results indeed from the assumption that the objects have constant complexity. The converse is also true: if we have a set of convex objects in general position, then their bisectors form an admissible system only if every pair of objects has exactly two supporting lines. Indeed, if this is not the case, one of the following holds : (1) the bisector is empty; (2) the bisector is homeomorphic to a ray; (3) there exist Voronoi cells that consist of more than one connected components.

Theorem 3 *Let $A = \{A_1, \dots, A_n\}$ be a set of smooth convex objects of constant complexity and in general position. The set of bisectors π_{ij} is an admissible system of bisectors iff every pair of objects has exactly two supporting lines.*

3 The Dynamic Algorithm

The algorithm that we propose is a variant of the randomized incremental algorithm for abstract Voronoi diagrams proposed by Klein and al. [6]. Our algorithm is fully dynamic and maintains the Voronoi diagram when a site is either added to the current set or deleted from it. To facilitate the presentation of the algorithm we first define the compactified version of the diagram and introduce the notion of conflict region.

The compactified diagram. We call 1-skeleton of the Voronoi diagram, the union of the Voronoi vertices and Voronoi edges. The 1-skeleton of the Voronoi diagram of an sc-pseudo-circles set $\mathcal{A}$ may consist of more than one connected components. However, we can define a compactified version of the diagram by adding to $\mathcal{A}$ a spurious site, A_∞ called the infinite site. The bisector of A_∞ and $A_i \in \mathcal{A}$ is a closed curve at infinity, intersecting any unbounded edge of the original diagram (see for example [5]). In the sequel we consider such a compactified version of the diagram, in which case the 1-skeleton is connected.

The conflict region. Each point x on a Voronoi edge incident to $V(A_i)$ and $V(A_j)$ is the center of a disk $C_{ij}(x)$ tangent to the boundaries ∂A_i and ∂A_j . This disk is called a *Voronoi bitangent disk*, and more precisely an interior Voronoi bitangent disk if it is included in $A_i \cap A_j$, or an exterior Voronoi bitangent disk if it lies in the complement of $A_i^\circ \cup A_j^\circ$. Similarly, a Voronoi vertex that belongs to the cells $V(A_i)$, $V(A_j)$ and $V(A_k)$ is the center of a disk $C_{ijk}(x)$ tangent to the boundaries of A_i , A_j and A_k . Such a disk is called a *Voronoi tritangent disk*, and more precisely an interior Voronoi tritangent disk if it is included in

$A_i \cap A_j \cap A_k$, or an external Voronoi tritangent disk if it lies in the complement of $A_i^\circ \cup A_j^\circ \cup A_k^\circ$.

Suppose we want to add a new object $A \notin \mathcal{A}$ and update the Voronoi diagram from $\mathcal{V}(\mathcal{A})$ to $\mathcal{V}(\mathcal{A}^+)$ where $\mathcal{A}^+ = \mathcal{A} \cup \{A\}$. We assume that $\mathcal{A}^+$ is also an sc-pseudo-circles set. The object A is said to be in conflict with a point x on the 1-skeleton of the current diagram if the Voronoi disk centered at x is either an internal Voronoi disk included in A° or an exterior Voronoi disk intersecting A° . We call *conflict region* the subset of the 1-skeleton of $\mathcal{V}(\mathcal{A})$ that is in conflict with the new object A . A Voronoi edge of $\mathcal{V}(\mathcal{A})$ is said to be in conflict with A if some part of this edge is in conflict with A .

Our dynamic algorithm relies on the two following theorems, which can be proved as in [6].

Theorem 4 *Let $\mathcal{A}^+ = \mathcal{A} \cup \{A\}$ be an sc-pseudo-circles set such that $A \notin \mathcal{A}$. The conflict region of A with respect to $\mathcal{V}(\mathcal{A})$ is a connected subset of the 1-skeleton of $\mathcal{V}(\mathcal{A})$.*

Theorem 5 *Let $\{A_i, A_j, A_k\}$ be an sc-pseudo-circles set in general position. Then the Voronoi diagram of A_i, A_j and A_k has at most two Voronoi vertices.*

Theorem 5 is equivalent to saying that two bisecting curves π_{ij} and π_{ik} relative to the same object A_i have at most two points of intersection. In particular, it implies that the conflict region of a new object A contains at most two connected subsets of each edge of $\mathcal{V}(\mathcal{A})$.

The data structures. The Voronoi diagram $\mathcal{V}(\mathcal{A})$ of the current set of objects is maintained through its dual graph $\mathcal{D}(\mathcal{A})$.

When a deletion is performed, a hidden site can reappear as visible. Therefore, we have to keep track of hidden sites. This is done through an additional data structure that we call the *covering graph* $\mathcal{K}(\mathcal{A})$. For each hidden object A_i , we call *covering set* of A_i a set $K(A_i)$ of objects such that any maximal disk included in A_i is included in the interior of at least one object of $K(A_i)$. In other words, in the Voronoi diagram $\mathcal{V}(K(A_i) \cup \{A_i\})$ the Voronoi cell $V(A_i)$ of A_i is empty. The covering graph is a directed acyclic graph with a node for each object. A node associated to a visible object is a root. The parents of a hidden object A_i are objects that form a covering set of A_i . The parents of a hidden object may be hidden or visible objects.

Note that if we perform only insertions or if it is known in advance that all sites will have non-empty Voronoi cells (e.g., this is the case for disjoint objects), it is not necessary to maintain a covering graph.

The algorithm needs to perform nearest neighbor queries. Optionally, the algorithm maintains a location data structure to perform efficiently those queries. The location data structure that we propose here is called a Voronoi hierarchy and described below in subsection 4.

3.1 The Insertion Procedure

The insertion of a new object A in the current Voronoi diagram $\mathcal{V}(\mathcal{A})$ involves the following steps: (1) find a first conflict between an edge of $\mathcal{V}(\mathcal{A})$ and A or

detect that A is hidden in $\mathcal{A}^+$; (2) find the whole conflict region of A ; (3) repair the dual graph; (4) update the covering graph; (5) update the location data structure if any. Steps 1 and 4 are discussed below. Steps 2 and 3 are performed exactly as in [9] for the case of disks. Briefly, Step 2 corresponds to finding the boundary of the star of A in $\mathcal{D}(\mathcal{A}^+)$. This boundary represents a hole in $\mathcal{D}(\mathcal{A})$, i.e., a sequence of edges of $\mathcal{D}(\mathcal{A})$ forming a topological circle. Step 3 simply amounts to “staring” this hole from A_i (which means to connect A_i to every vertex on the hole boundary).

Finding the first conflict or detecting a hidden object. The first crucial operation to perform when inserting a new object is to determine if the inserted object is hidden or not. If the object is hidden we need to find a covering set of this object. If the object is not hidden we need to find an edge of the current diagram in conflict with the inserted object.

The detection of the first conflict is based on closest site queries. Such a query takes a point x as input and asks for the object in the current set $\mathcal{A}$ that is closest to x . If we didn’t have any location data structure, then we perform the following *simple walk* on the Voronoi diagram to find the object in $\mathcal{A}$ closest to x . The walk starts from any object $A_i \in \mathcal{A}$ and compares the distance $\delta(x, A_i)$ with the distances $\delta(x, A)$ to the neighbors A of A_i in the Voronoi diagram $\mathcal{V}(\mathcal{A})$. If some neighbor A_j of A_i is found closer to x than A_i , the walk proceeds to A_j . If there is no neighbor of A_i that is closer to x than A_i , then A_i is the object closest to x among all objects in $\mathcal{A}$. It is easy to see that this walk can take linear time. We postpone until the next section the description of the location data structure and the way these queries can be answered more efficiently.

Let us consider first the case of disjoint objects. In this case there are no hidden objects and each object is included in its own cell. We perform a closest site query for any point p of the object A to be inserted. Let A_i be the object of $\mathcal{A}$ closest to p . The cell of A_i will shrink in the Voronoi diagram $\mathcal{V}(\mathcal{A}^+)$ and at least one edge of $\partial V(A_i)$ is in conflict with A . Hence, we only have to look at the edges of $\partial V(A_i)$ until we find one in conflict with A .

When objects do intersect, we perform an operation called *location of the medial axis*, which either provides an edge of $\mathcal{V}(\mathcal{A})$ that is in conflict with A , or returns a covering set of A . There is a simple way to perform this operation. Indeed, the medial axis $S(A)$ of A is a tree embedded in the plane, and for each object A_i , the part of $S(A)$ that is not covered by A_i (that is the part of $S(A)$ made up by the centers of maximal disks in A , not included in A_i) is connected. We start by choosing a leaf vertex p of the medial axis $S(A)$ and locate the object A_i that is closest to p . Then we prune the part of the medial axis covered by A_i and continue with the remainder of the medial axis in exactly the same way. If, at some point, there is no part of $S(A)$ left, we know that A is hidden, and the set of objects A_i , which pruned a part of $S(A)$, forms a covering of A . Otherwise we perform a nearest neighbor query for any point of $S(A)$ which has not been pruned. A first conflict can be found from the answer to this query in exactly the same way as in the case of disjoint objects, discussed above.

It remains to explain how we choose the objects A_i that are candidates for covering parts of $S(A)$. As described above, we determine the first object A_i by performing a nearest neighbor query for a leaf vertex p of $S(A)$. Once we have pruned the medial axis, we consider one of the leaf vertices p' created after the pruning. This corresponds to a maximal circle $M(p')$ of A centered at p' , which is also internally tangent to A_i . To find a new candidate object for covering $S(A)$, we simply need to find a neighbor of A_i in the Voronoi diagram that contains $M(p')$; if $M(p')$ is actually covered by some object in $\mathcal{A}$, then it is guaranteed that we will find one among the neighbors of A_i . We then continue, as above, with the new leaf node of the pruned medial axis and the new candidate covering object, as above.

Updating the covering graph. We now describe how Step 4 of the insertion procedure is performed. We start by creating a node for A in the covering graph.

If A is hidden, the location of its medial axis yields a covering set $K(A)$ of A . In the covering graph we simply assign the objects in $K(A)$ as parents of A .

If the inserted object A is visible, some objects in $\mathcal{A}$ can become hidden due to the insertion of A . The set of objects that become hidden because of A are provided by Step 2 of the insertion procedure. They correspond to cycles in the conflict region of A . The main idea for updating the covering graph is to look at the neighbors of A in the new Voronoi diagram.

Lemma 6 *Let $\mathcal{A}$ be an sc-pseudo-circles set. Let $A \notin \mathcal{A}$ be an object such that $\mathcal{A}^+ = \mathcal{A} \cup \{A\}$ is also an sc-pseudo-circles set and A is visible in $\mathcal{V}(\mathcal{A}^+)$. If an object $A_i \in \mathcal{A}$ becomes hidden upon the insertion of A , then the neighbors of A in $\mathcal{V}(\mathcal{A}^+)$ along with A is a covering set of A_i .*

Let A_i be an object that becomes hidden upon the insertion of A . By Lemma 6 the set of neighbors of A in $\mathcal{V}(\mathcal{A}^+)$ along with A is a covering set $K(A_i)$ of A_i . The only modification we have to do in the covering graph is to assign all objects in $K(A_i)$ as parents of A_i .

Updating the location data structure. The update of the location data structure is really simple. Let A be the object inserted. If A is hidden we do nothing. If A is not hidden, we insert A in the location data structure, and delete from it all objects that become hidden because of the insertion of A .

3.2 The Deletion Procedure

Let A_i be the object to be deleted and let $K_p(A_i)$ be the set of all objects in the covering graph $\mathcal{K}(\mathcal{A})$ that have A_i as parent. The deletion of A_i involves the following steps: (1) remove A_i from the dual graph; (2) remove A_i from the covering graph; (3) remove A_i from location data structure; (4) reinsert the objects in $K_p(A_i)$.

Step 1 requires no action if A_i is hidden. If A_i is visible, we first build an annex Voronoi diagram for the neighbors of A_i in $\mathcal{V}(\mathcal{A})$ and use this annex Voronoi diagram to fill in the cell of A_i (see [9]). In Step 2, we simply delete all edges of $\mathcal{K}(\mathcal{A})$ to and from A_i , as well as the node corresponding to A_i . In

Step 3, we simply delete A_i from the location data structure. Finally, in Step 4 we apply the insertion procedure to all objects in $K_p(A_i)$. Note, that if A_i is hidden, this last step simply amounts to finding a new covering set for all objects in $K_p(A_i)$.

4 Closest Site Queries

The location data structure is used to answer closest site queries. A closest site query takes as input a point x and asks for the object in the current set $\mathcal{A}$ that is closest to x . Such queries can be answered through a simple walk in the Voronoi diagram (as described in the previous section) or using a hierarchical data structure called the Voronoi hierarchy.

The Voronoi hierarchy. The hierarchical data structure used here, denoted by $\mathcal{H}(\mathcal{A})$, is inspired from the Delaunay hierarchy proposed by Devillers [10]. The method consists of building the Voronoi diagrams $\mathcal{V}(\mathcal{A}_\ell)$, $\ell = 0, \dots, L$, of a hierarchy $\mathcal{A} = \mathcal{A}_0 \supseteq \mathcal{A}_1 \supseteq \dots \supseteq \mathcal{A}_L$ of subsets of $\mathcal{A}$. Our location data structure conceptually consists of all subsets $\mathcal{A}_\ell$, $1 \leq \ell \leq L$.

The hierarchy $\mathcal{H}(\mathcal{A})$ is built together with the Voronoi diagram $\mathcal{V}(\mathcal{A})$ according to the following rules. Any object of $\mathcal{A}$ is inserted in $\mathcal{V}(\mathcal{A}_0) = \mathcal{V}(\mathcal{A})$. If A has been inserted in $\mathcal{V}(\mathcal{A}_\ell)$ and is visible, it is inserted in $\mathcal{V}(\mathcal{A}_{\ell+1})$ with probability β . If, upon the insertion of A in $\mathcal{V}(\mathcal{A})$, an object becomes hidden it is deleted from all diagrams $\mathcal{V}(\mathcal{A}_{\ell'})$, $\ell' > 0$, in which it has been inserted. Finally, when an object A_i is deleted from the Voronoi diagram $\mathcal{V}(\mathcal{A})$, we delete A_i from all diagrams $\mathcal{V}(\mathcal{A}_\ell)$, $\ell \geq 0$, in which it has been inserted. Note that all diagrams $\mathcal{V}(\mathcal{A}_\ell)$, $\ell > 0$, do not contain any hidden objects.

The closest site query for a point x is performed as follows. The query is first performed in the top-most diagram $\mathcal{V}(\mathcal{A}_L)$ using the simple walk. Then, for $\ell = L - 1, \dots, 0$ a simple walk is performed in $\mathcal{V}(\mathcal{A}_\ell)$ from $\mathcal{A}_{\ell+1}$ to $\mathcal{A}_\ell$ where $\mathcal{A}_{\ell+1}$ (resp. $\mathcal{A}_\ell$) is the object of $\mathcal{A}_{\ell+1}$ (resp. of $\mathcal{A}_\ell$) closest to x .

It is easy to show that the expected size of $\mathcal{H}(\mathcal{A})$ is $O(\frac{1}{1-\beta} n)$, and that the expected number of levels in $\mathcal{H}(\mathcal{A})$ is $O(\log_{1/\beta} n)$. Moreover, it can be proved that the expected number of steps performed by the walk at each level is constant ($O(1/\beta)$).

We still have to bound the time spent in each visited cell. Let A_i be the site of a visited cell in $\mathcal{V}(\mathcal{A}_\ell)$. Because the complexity of any cell in a Voronoi diagram is only bounded by $O(n_\ell)$ if n_ℓ is the number of sites, it is not efficient to compare the distances $\delta(x, A_i)$ and $\delta(x, A)$ for each neighbor A of A_i in $\mathcal{V}(\mathcal{A}_\ell)$. Therefore we attach an additional balanced binary tree to each cell of each Voronoi diagram in the hierarchy. The tree attached to the cell $V_\ell(A_i)$ of A_i in the diagram $\mathcal{V}(\mathcal{A}_\ell)$ includes, for each Voronoi vertex v of $V_\ell(A_i)$, the ray $\rho_i(p_v)$ where p_v is the point on ∂A_i closest to v , and $\rho_i(p_v)$ is defined as the ray starting from the center of the maximal disk $M_i(p_v)$ and passing through p_v . The rays are sorted according to the (counter-clockwise) order of the points p_v on ∂A_i . When $V_\ell(A_i)$ is visited, the ray $\rho_i(p_x)$ corresponding to the query point x is localized using the tree. Suppose that it is found to be between the rays of

two vertices v_1 and v_2 . Then it suffice to compare $\delta(x, A_i)$ and $\delta(x, A_j)$ where A_j is the neighbor of A_i in $\mathcal{V}(\mathcal{A}_\ell)$ sharing the vertices v_1 and v_2 . Thus the time spend in each visited cell of $\mathcal{V}(\mathcal{A}_\ell)$ is $O(\log n_\ell) = O(\log n)$, which (together with with the expected number of visited nodes) yields the following lemma

Lemma 7 *Using a hierarchy of Voronoi diagrams with additional binary trees for each cell, a closest site query can be answered in time $O(\frac{1}{\beta \log(1/\beta)} \log^2 n)$.*

5 Complexity Analysis

In this section we deal with the cost of the basic operations of our dynamic algorithm. We consider three scenarios. The first one assumes objects do not intersect. In the second scenario objects intersect but there are no hidden objects. The third scenario differs from the second one in that we allow the existence of hidden objects.

In each of the above three cases, we consider the expected cost of the basic operations, namely insertion and deletion. The expectation refers to the insertion order, that is, all possible insertion orders are considered to be equally likely and each deletion is considered to deal equally likely with any object in the current set. In all cases we assume that the Voronoi diagram hierarchy is used as the location data structure. Note that the hierarchy introduces another source of randomization. In the first two scenarios, i.e., when no hidden object exist, there is no covering graph to be maintained. Note the the randomized analysis obviously does not apply to the reinsertion of objects covered by a deleted object A_i , which explains why the randomization fails to improve the complexity of deletion in the presence of hidden objects.

Our results are summarized in the table below. The corresponding proofs are omitted due to lack of space; in any case they follow directly from a careful step by step analysis of the insertion and deletion procedures described above.

	Disjoint	No hidden	Hidden
Insertion	$O(\log^2 n)$	$O(n)$	$O(n)$
Deletion	$O(\log^3 n)$	$O(n)$	$O(n^2)$

6 Extensions

In this section we consider several extensions of the problem discussed in the preceding sections.

Degenerate configurations. Degenerate configurations occur when the set contains pairs of internally tangent objects. Let $\{A_i, A_j\}$ be an sc-pseudo-circles set with A_i and A_j internally tangent and $A_i \subseteq A_j$. The bisector π_{ij} is homeomorphic to a ray, if A_i and A_j have a single tangent point, or to two disconnected rays, if A_i and A_j have two tangent points. In any case, the interior $V^\circ(A_i)$ of the Voronoi region of A_i in $\mathcal{V}(\{A_i, A_j\})$ is empty and we consider the object A_i

as hidden. This point of view is consistent with the definition we gave for hidden sites, which is that an object A is hidden if $N^\circ(A) = \emptyset$.

Let us discuss the algorithmic consequences of allowing degenerate configurations. When the object A is inserted in the diagram, the case where A is internally tangent to a visible object $A_i \in \mathcal{A}$ is detected at Step 1, during the location the medial axis of A . The case of an object $A_j \in \mathcal{A}$ is internally tangent to A is detected during Step 2, when the entire conflict region is searched. In the first case A is hidden and its covering set is $\{A_i\}$. In the second case A_i becomes hidden and its covering set is $\{A\}$.

Pseudo-circles sets of piecewise smooth convex objects. In the sections above we assumed that all convex objects have smooth boundaries, i.e., their boundaries are at least C^1 -continuous. In fact we can handle quite easily the case of objects whose boundaries are only piecewise C^1 -continuous. Let us call *vertices* the points on the boundary of an object where there is no C^1 -continuity. The main problem of piecewise C^1 -continuous objects is that they can yield two-dimensional bisectors when two objects share the same vertex. The remedy is similar to the commonly used approach for the Voronoi diagram of segments (e.g., cf. [11]): we consider the vertices on the boundary of the objects as objects by themselves and slightly change the distance so that a point whose closest point on object A_i is a vertex of A_i is considered to be closer to that vertex. All two-dimensional bisectors then become the Voronoi cells of these vertices.

As far as our basic operations are concerned, we proceed as follows. Let A be the object to be inserted or deleted. We note A_v the set of vertices of A and $\hat{A}$ the object A minus the points in A_v . When we want to insert A in the current Voronoi diagram we at first insert all points in A_v and then $\hat{A}$. When we want to delete A we at first delete $\hat{A}$ and then all points in A_v . During the latter step we have to make sure that points in A_v are not vertices of other objects as well. This can be done easily by looking at the neighbors in the Voronoi diagram of each point in A_v .

Generic convex objects. In the case of smooth convex objects which do not form pseudo-circles sets we can compute the Voronoi diagram in the complement of their union (free space). The basic idea is that the Voronoi diagram in free space depends only on the arcs appearing on the boundary of the union of the objects.

More precisely, let $\mathcal{A}$ be a set of convex objects and let C be a connected component of the union of the objects in $\mathcal{A}$. Along the boundary ∂C of C , there exists a sequence of points $\{p_1, \dots, p_m\}$, which are points of intersection of objects in $\mathcal{A}$. An arc α_i on ∂C joining p_i to p_{i+1} belongs to a single object $A \in \mathcal{A}$. We form the piecewise smooth convex object A_{α_i} , whose boundary is $\alpha_i \cup p_i p_{i+1}$, where $p_i p_{i+1}$ is the segment joining the points p_i and p_{i+1} . Consider the set $\mathcal{A}'$ consisting of all such objects A_{α_i} . $\mathcal{A}'$ is a pseudo-circles set (consisting of disjoint piecewise smooth convex objects) and the Voronoi diagrams $\mathcal{V}(\mathcal{A})$ and $\mathcal{V}(\mathcal{A}')$ coincide in free space.

The set $\mathcal{A}'$ can be computed by performing a line-sweep on the set $\mathcal{A}$ and keeping track of the boundary of the connected components of the union of the

objects in $\mathcal{A}$. This can be done in time $O(n \log n + k)$, where $k = O(n^2)$ is the complexity of the boundary of the afore-mentioned union. Since the objects in $\mathcal{A}'$ are disjoint, we can then compute the Voronoi diagram in free space in total expected time $O(k \log^2 n)$.

7 Conclusion

We presented a dynamic algorithm for the construction of the euclidean Voronoi diagram in the plane for various classes of convex objects. In particular, we considered pseudo-circles sets of piecewise smooth convex objects, as well as generic smooth convex objects, in which case we can compute the Voronoi diagram in free space. Our algorithm uses fairly simple data structures and enables us to perform deletions easily.

We are currently working on extending the above results to non-convex objects, as well as understanding the relationship between the euclidean Voronoi diagram of such objects and abstract Voronoi diagrams. We conjecture that, given a pseudo-circles set in general position, such that any pair of objects has exactly two supporting lines, the corresponding set of bisectors is an admissible system of bisectors.

References

1. Aurenhammer, F., Klein, R.: Voronoi diagrams. In Sack, J.R., Urrutia, J., eds.: *Handbook of Computational Geometry*. Elsevier Science Publishers B.V. North-Holland, Amsterdam (2000) 201–290
2. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N.: *Spatial tessellations: concepts and applications of Voronoi diagrams*. 2nd edn. John Wiley & Sons Ltd., Chichester (2000)
3. Koltun, V., Sharir, M.: Polyhedral Voronoi diagrams of polyhedra in three dimensions. In: *Proc. 18th Annu. ACM Sympos. Comput. Geom.* (2002) 227–236
4. Koltun, V., Sharir, M.: Three dimensional euclidean Voronoi diagrams of lines with a fixed number of orientations. In: *Proc. 18th Annu. ACM Sympos. Comput. Geom.* (2002) 217–226
5. Klein, R.: *Concrete and Abstract Voronoi Diagrams*. Volume 400 of *Lecture Notes Comput. Sci.* Springer-Verlag (1989)
6. Klein, R., Mehlhorn, K., Meiser, S.: Randomized incremental construction of abstract Voronoi diagrams. *Comput. Geom.: Theory & Appl.* **3** (1993) 157–184
7. Alt, H., Schwarzkopf, O.: The Voronoi diagram of curved objects. In: *Proc. 11th Annu. ACM Sympos. Comput. Geom.* (1995) 89–97
8. McAllister, M., Kirkpatrick, D., Snoeyink, J.: A compact piecewise-linear Voronoi diagram for convex sites in the plane. *Discrete Comput. Geom.* **15** (1996) 73–105
9. Karavelas, M.I., Yvinec, M.: Dynamic additively weighted Voronoi diagrams in 2D. In: *Proc. 10th Europ. Sympos. Alg.* (2002) 586–598
10. Devillers, O.: The Delaunay hierarchy. *Internat. J. Found. Comput. Sci.* **13** (2002) 163–180
11. Burnikel, C.: *Exact Computation of Voronoi Diagrams and Line Segment Intersections*. Ph.D thesis, Universität des Saarlandes (1996)

Buffer Overflows of Merging Streams

Alex Kesselman¹, Zvi Lotker², Yishay Mansour³, and Boaz Patt-Shamir^{4*}

¹ School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
alx@cs.tau.ac.il

² Dept. of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel.
zvilo@eng.tau.ac.il

³ School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
mansour@cs.tau.ac.il

⁴ Cambridge Research Lab, Hewlett-Packard, One Cambridge Center, Cambridge, MA 02142.
Boaz.PattShamir@HP.com

Abstract. We consider a network merging streams of packets with different quality of service (QoS) levels, where packets are transported from input links to output links via multiple merge stages. Each merge node is equipped with a finite buffer, and since the bandwidth of a link outgoing from a merge node is in general smaller than the sum of incoming bandwidths, overflows may occur. QoS is modeled by assigning a positive value to each packet, and the goal of the system is to maximize the total value of packets transmitted on the output links. We assume that each buffer runs an independent local scheduling policy, and analyze FIFO policies that must deliver packets in the order they were received. We show that a simple local on-line algorithm called Greedy does essentially as well as the combination of locally optimal (off-line) schedules. We introduce a concept we call the *weakness* of a link, defined as the ratio between the longest time a packet spends in the system before transmitted over the link, and the longest time a packet spends in that link's buffer. We prove that for any tree, the competitive factor of Greedy is at most the maximal link weakness.

1 Introduction

Consider an Internet service provider (ISP), or a corporate intranet, that connects a large number of users with the Internet backbone using an “uplink.” Within such a system, consider the traffic oriented towards the uplink, namely the streams whose start points are the local users and whose destinations are outside the local domain. Then streams are merged by a network that consists of *merge nodes*, typically arranged in a tree topology whose root is directly connected to the uplink. Without loss of generality, we may assume that the bandwidth of the link emanating from a merge node is less than the sum of bandwidths of incoming links (otherwise, we can assume that the incoming links are connected directly to the next node up). Hence, when all users inject data at maximum local speed, packets will eventually be discarded. A very effective way to mitigate some of the losses due to temporary overloads is to equip the merge nodes with buffers, that can absorb transient bursts by storing incoming packets while the outgoing link is busy.

* On leave from Dept. of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel.

The merge nodes are controlled by local on-line *buffer management algorithms* whose job is to decide which packets to forward and which to drop so as to minimize the damage in case of an overflow.

In this paper we study the performance of various buffer management algorithms in the context of a system of merging streams, under the assumption that the system is required to support different quality of service (QoS) levels. The different QoS levels are modeled by assuming that each packet has a positive value, and that the goal of the system is to maximize the total value of packets delivered.

Evaluating the performance of the system cannot be done in absolute terms, since the total value delivered depends on the actual streams that arrive. Instead, we measure the competitive ratio of the algorithm [18] by bounding, over all possible input sequences, the ratio between the value gained by the algorithm in question, and the best possible value that can be gained by any schedule.

Our model. To allow us to describe our results, let us give here a brief informal overview of the model (more details are provided in Section 2). Our model is essentially the model used by Adversarial Queuing Theory [5], with the following important differences: packet injection is unrestricted, buffers are finite, and each packet has a value. More specifically, the system is described by a communication graph, where each link e has a buffer Q_e in its ingress and a prescribed bandwidth $W(e)$. An execution of the system proceeds in synchronous steps. In each step, new packets may enter the system, where each packet has a value (in $\mathbb{R}^+$), and a completely specified route. Also in each step, packets may progress along edges, some packets may be dropped from the system, and some packets may be absorbed by their destinations. The basic limitation on these actions is that for each edge e , at most $W(e)$ packets may cross it in each step, and at most $\text{size}(Q_e)$ packets may be retained in the buffer from step to step. The task of the buffer management algorithm is to decide which packets to forward and which packets to drop subject to these restrictions. Given a system and an input sequence, the total value of a schedule for that input is the total value of the packets that reach their destinations.

In this paper, we consider a few special cases of the general model above, justified by practical engineering considerations. The possible restrictions are on the network topology, scheduling algorithms, and packet values. The variants are as follows. *Tree topology* assumes that the union of the paths of all packets is a directed tree, where all paths start from a leaf and end at the root of the tree. Regarding schedules, our results are for the class of *work-conserving* schedules, i.e., schedules that always forward a packet when the buffer is non-empty [9].¹ We consider the class of FIFO algorithms, i.e., algorithms that may not send a packet that arrives late before a packet that arrives early. This condition is natural for many network protocols (e.g., TCP).

Our results. We study the effect of different packet values, different buffer sizes and link bandwidths on the competitiveness of various local algorithms. We study very simple Greedy algorithm that drops the least valuable packets available when there is an overflow. We also consider the Locally Optimal schedule, which is the best possible schedule with respect to a single buffer. Roughly speaking, it turns out that in many

¹ Work conserving schedules are sometimes called “greedy” [16,5]. In line with the networking community, we use the term “work conserving” here; we reserve the term “greedy” for a specific algorithm we specify later.

cases, the Greedy algorithm has performance which is asymptotically equivalent to the performance of a system defined by a composition of locally optimal schedules, and in some cases, its performance is proportional to the global optimum. More specifically, we obtain the following results.

First, we present simple scenarios that show that local algorithms cannot be too good: specifically, even allowing each node to run the locally optimal (offline) schedule may result in competitive ratio of $\Omega(h)$ on height- h trees with uniform buffer sizes and uniform link bandwidths. For bounded degree trees of height h , the competitive factor drops to $\Omega(h/\log h)$, and for trees of height h and $O(h)$ nodes, the lower bound drops further to $\Omega(\sqrt{h})$.

Next, we analyze the Greedy algorithm. By extending the analysis of the single buffer case, we show that for arbitrary topologies, the maximal ratio between the performance of Greedy and the performance of any work-conserving (off-line) schedule is $O(DR/B_{\min})$, where D is the length of the longest packet route (measured in time units), R is the maximal rate in which packets may reach their destinations, and $B_{\min}$ is the size of the smallest buffer in the system.

We then focus on tree topologies, where we present our most interesting result. We introduce the concept of *link weakness*, defined as follows. For any given link e , define the *delay* of e to be the longest time a packet can spend in the buffer of e (for work-conserving schedules, it's exactly the buffer size divided by the link bandwidth). Define further the *height* of e to be the maximal length of a path from an input leaf to the egress of e , where the length of a link is its delay. Finally, the *weakness* of e , denoted $\lambda(e)$, is the ratio between its height and its delay (we have that $\lambda(e) \geq 1$). Our main result is that the competitive factor of Greedy is proportional to the maximal link weakness in the system. Our proof is for the case where each packet has one of two possible values.

Related work. There is a myriad of research papers about packet drop policies in communication networks—see, e.g., the survey of [13] and references therein. Some of the drop mechanisms (most notably RED [7]) are designed to signal congestion to the sending end. The approach abstracted in our model is implicit in the recent DiffServ model [4,6] and ATM [19].

There has been work on analyzing various aspects of this model using classical queuing theory, and assuming Poisson arrivals [17]. The Poisson arrival model has been seriously undermined by recent discoveries regarding the nature of traffic in computer networks (see, e.g., [14,20]).

In this work we use competitive analysis, which studies the worst-case performance guarantees of an on-line algorithm relative to an off-line solution. This approach is used in Adversarial Queuing Theory [5], where packet injections are restricted, and the main measure of performance is the size of the buffers required to never drop any packet. In a recent paper, Aiello et al. [1] propose to study the *throughput* of a network with bounded buffers and packet drops. Their model is similar to ours, so let us point out the differences. The model of [1] assumes uniform buffer sizes, link bandwidths, and packet values, whereas we consider individual sizes, bandwidths and values. As we show in this paper, these factors have a decisive effect on the competitiveness of the system even in very simple cases. Another difference is that [1] compares on-line algorithms to any off-line schedule, including ones that are not work-conserving. Due

to this approach, the performance guarantees they can prove are rather weak, and thus they are mainly interested in whether the competitive factor of a scheduling policy is finite or not. By contrast, we consider work-conserving off-line schedules, which allow us to derive quantitative results and gain more insights from the practical point of view.

Additional relevant references study the performance guarantees of a single buffer, where packets have different values. The works of [2,12] study the case where one cannot preempt a packet already in the buffer. In [10], an upper bound of 2 is proven for the competitive factor of the greedy algorithm. The two-value single buffer case is further studied in [11,15]. Overflows in a shared-memory switch are considered in [8].

A recent result of Azar and Richter [3] analyzes a scenario of stream merging in input-queued switches. Briefly, finite buffers are located at input ports; the output port has no buffer: it selects, at each step, one of the input buffers and transmits the packet in the head of that buffer. Their main result is a centralized algorithm that reduces this scenario of a single merge to the problem of managing a single buffer, while incurring only a constant blowup in the competitive factor.

Paper organization. Section 2 contains the model description. Lower and upper bounds for local schedules are considered in Section 3 and Section 4, respectively.

2 Model and Notation

We start with a description of the general model.

The system is defined by a directed graph $G = (V, E)$, where each link $e \in E$ has *bandwidth* (or *speed*) $W(e) \in \mathbb{N}$, and a *buffer* Q_e with storage capacity $size(Q_e) \in \mathbb{N} \cup \{0\}$. (The buffer resides at the link's ingress—see below.)

The input to the system is a sequence of *packet injections*, one for each time step. A packet injection is a set of packets, where each packet p is characterized by its route, denoted $route(p)$, and its value, denoted $\omega(p)$.² The first node on the route is called the packet's *source*, and the last node is called the packet's *destination*. To avoid trivialities, we assume that each packet route is a simple path that contains at least one link.

The *execution* (or *schedule*) of the system proceeds in synchronous steps as follows. The state of the system is defined by the current contents of each link's buffer Q_e , and by each link's *transit contents*, denoted $transit_e$ for a link e . Initially, all buffers and transit contents are empty sets. Each step consists of the following substeps.

- (1) **Packet injection:** For each link e , an arbitrary set of new packets whose first link is e is added to Q_e .
- (2) **Packet delivery:** For all links $e_1 = (u, v)$ and $e_2 = (v, w)$, all packets currently in $transit_{e_1}$ whose next route edge is e_2 are moved from $transit_{e_1}$ into Q_{e_2} . All packets whose destination is v are *absorbed*. After this substep, $transit_e = \emptyset$ for all $e \in E$.
- (3) **Packet drop:** A subset of the packets currently stored in Q_e is removed from Q_e , for each $e \in E$.
- (4) **Packet send:** For each link e , a subset of the packets currently stored in Q_e is moved from Q_e to $transit_e$.

² There may be many packets with the same route and value, so technically each packet injection is a multiset; we abuse notation slightly, and always refer to multisets when we say “sets.”

We stress that packet injection rate is unrestricted (as opposed, e.g., to Adversarial Queuing Theory). Note also that we assume that all link latencies are one unit.

A *scheduling algorithm* determines which packets to drop (Substep 3) and which packets to send (Substep 4), so as to satisfy the following conditions after each step is completely done:

- For each link e , the number of packets stored in Q_e is at most $size(Q_e)$.³
- For each link e , the total number of packets stored in the transit contents of e is at most $W(e)$.

Given an input sequence $\mathcal{I}$ and an algorithm A for a system, the *value* obtained by A for $\mathcal{I}$, denoted $\omega_A(\mathcal{I})$, is the sum of values of all packets that have reached their destination.

Tree Topology. A system is said to have *tree topology* if the union of all packet routes used in the system is a tree, where packet sources are leaves and all packets are destined at the single root. In this case each node except the root has a single storage buffer (associated with its unique outgoing edge), sometimes referred to as the *node's* buffer. It is convenient also to assume in the tree case that the leaves and root are links: this way, we have streams entering the system and a stream leaving the system. We say that a node v is *upstream* from u (or, equivalently, u is *downstream* from v), if there is a directed path from v to u .

FIFO Schedules. We consider FIFO schedules, which adhere to the rule that packets are sent over a link in the same order they enter the buffer at the tail of the link (packets may be arbitrarily dropped by the algorithm, but the packets that do get sent preserve their relative order). More precisely, for all packets p, q and every link e : If p is sent on e at time t and q is sent on e at time $t' > t$, then q did not enter Q_e before p .

Work-Conserving Schedules. A given schedule is called *work conserving* if for every step t and every link e we have that the number of packets sent over e at step t is the minimum between $W(e)$ and the number of packets in Q_e (at step t just before Substep 4). Intuitively, a work conserving schedule always forwards the maximal number of packets allowed by the local bandwidth restriction. (Note that packets may be dropped in a work-conserving schedule even if the buffer is not full.)

Algorithms and Their Evaluation. An algorithm is called *local on-line* if its action at time t at node v depends only on the sequence of packets arriving at v up to time t . An algorithm is called *local off-line* if its action at time t at node v depends only on the sequence of packets arriving at v , including packets that arrive at v after t . Given a sequence of packet arrivals and injections at node v , the local-offline schedule with the maximum output value of v for the given sequence is the Local Optimal schedule, denoted $OptL_v$. When the set of routes is acyclic, we define the schedule $OptL$ to be the composition of Local Optimal schedules, constructed by applying $OptL_v$ in topological order. A *global off-line* schedule has the whole input (at all nodes, at all times) available ahead of any decision. We denote by Opt the global off-line work-conserving schedule with the maximum value.

Given a system and an algorithm A for that system, the *competitive ratio* (or *competitive factor*) of A is the worst-case ratio, over all input sequences, between the value

³ Note that the restriction applies only *between* steps: in our model, after Substeps 1,2 and before Substeps 3,4, more than $size(Q_e)$ packets may be stored in Q_e .

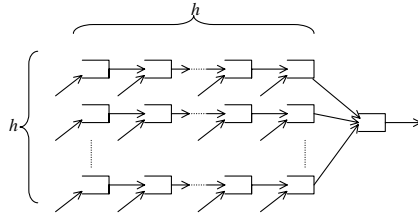


Fig. 1. Topology used in the proof of Theorem 1, with parameter h . Diagonal arrows represent input links, and the rightmost arrow represents the output link.

of Opt and the value of A . Formally:

$$\text{cr}(A) = \sup \left\{ \frac{\omega_{\text{Opt}}(\mathcal{I})}{\omega_A(\mathcal{I})} : \mathcal{I} \text{ is an input sequence} \right\}.$$

Since we deal with a maximization problem this ratio will always be at least 1.

3 Lower Bounds for Local Schedules

In this section we consider simple scenarios that establish lower bounds on local algorithms. We show that even if each node runs OptL – a locally optimal schedule (that may be computed off-line) – the performance cannot be very close to the globally optimal schedule.

As we are dealing with lower bounds, we will be interested in very simple settings. In the scenarios below, all buffers have the same size B and all links have bandwidth 1. Furthermore, we use only two packet values: low value of 1, and high value of $\alpha > 1$. (The bounds of Theorems 2 and 3 are tight for the two-value case; we omit details here.)

As an immediate corollary of Theorem 4, we have that the lower bound of Theorem 1 is tight, as argued below.

Theorem 1. *The competitive ratio of OptL for a tree-topology system is $\Omega(\min(h, \alpha))$, where h is the depth of the tree.*

Proof: Consider a system with $h^2 + 1$ nodes, where h^2 “path nodes” have input links, and are arranged in h paths of length h each, and one “output node” has input from the h last path nodes, and has one output link (see Figure 1). Let B denote the size of a buffer. The input sequence is as follows. The input for all nodes in the beginning of a path is B packets of value α followed by B packets of value 1 (at steps $0, \dots, 2B - 1$). The input for the i -th node on each path for $i > 1$ is B packets of value 1 at time $B(i - 2) + i - 1$.

Consider the schedule of OptL first. There are no overflows on the buffers of the path nodes, and hence it is easy to verify by induction that the output from the i -th node on any path contains $B \cdot i$ packets of value 1, followed by B packets of value α . Thus, the output node gets h packets of value 1 in each time step t for $t = h, \dots, h \cdot B$, and h packets of value α in each time step t for $t = h \cdot B + 1, \dots, (h + 1) \cdot B + 1$. Clearly, the value of OptL in this case consists of $(h - 1)B$ low value packets and $2B$ high value packets.

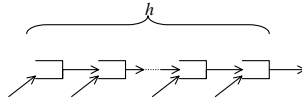


Fig. 2. A line of depth h . Diagonal arrows represent input links, and the rightmost arrow represents the output link.

On the other hand, the globally optimal schedule Opt is as follows. On the j -th path, the first $B(j-1)$ low value packets are dropped. Thus, the stream outcoming from the j -th path consists of $B(h-(j-1))$ low value packets followed by B high value packets, so that in each time step $t = h, \dots, hB$ exactly one high value packet and $h-1$ low value packets enter the output node, and Opt obtains the total value of $hB\alpha + B$. It follows that the competitive ratio of OptL in this case is $\frac{h\alpha+1}{(h-1)+2\alpha} = \Omega\left(\frac{h\alpha}{h+\alpha}\right) = \Omega(\min(h, \alpha))$.

■

If we insist on bounded-degree trees, the above lower bound changes slightly, as stated below. The proof is omitted from this extended abstract.

Theorem 2. *The competitive ratio of OptL for a binary tree with depth h is $\Theta(\min(\alpha, \frac{h}{\log h}))$.*

Further restricting attention to a line topology (see Figure 2), the lower bound for $\alpha \gg h$ decreases more significantly, as the following result shows. Proof is omitted.

Theorem 3. *The competitive ratio of OptL for a line of length h is $\Theta(\min(\alpha, \sqrt{h}))$.*

4 Upper Bounds for Local Schedules

In this section we study the competitive factor of local schedules. We first prove a simple upper bound for arbitrary topology, and then give our main result which is an upper bound for the tree topology.

4.1 An Upper Bound on Greedy Schedules for General Topology

We now turn to positive results, namely upper bounds on the competitive ratio of a natural on-line local algorithm [10].

Algorithm 1 Greedy: *Never discard packets if there is free storage space. When an overflow occurs, drop the packets of the least value.*

We now prove an upper bound on the competitiveness of Greedy in general topologies. We remark that all lower bounds proved in Section 3 for OptL hold also for Greedy as well (details omitted).

We start with the following basic definition.

Definition 1. *For a given link e in a given system, we define the delay of e , denoted $d(e)$, to be the ratio $\lceil \text{size}(Q_e) / W(e) \rceil$. The delay of a given path is the sum of the edge delays on that path. The maximal delay in a system, denoted D , is the maximal delay over all simple paths in the systems.*

Note that the delay of a buffer is the maximal number of time units a packet can be stored in it under any work-conserving schedule.

We also use the concept of drain rate, which is the maximal possible rate of packet absorption. Formally, it is defined as follows.

Definition 2. Let Z be the set of all links leading to an output node in a given system. The drain rate of the system, denote R , is the sum $\sum_{e \in Z} W(e)$.

With these notions, we can now state and prove the following general result. Note that the result is independent of node degrees.

Theorem 4. For any system with maximal delay at most D , drain rate at most R , and buffers with size at least $B_{\min}$, the competitive ratio of Greedy is $O(DR/B_{\min})$.

We remark that the proof given below holds also for OptL.

Proof: Fix an input sequence $\mathcal{I}$. Divide the schedule into time intervals $I_j = [jD, (j+1)D - 1]$ D time steps each. Consider a time interval I_j . Define S_j to be the set of $2DR$ most valuable packets that are injected into the system during I_j . Observe that in a work conserving schedule, any packet is either absorbed or dropped in D time units. It follows that among all packets that arrive in I_j , at most $2DR$ will be eventually absorbed by their destinations: DR may be absorbed during I_j , and DR during the next interval of D time units (i.e. I_{j+1}). Since this property holds for any work-conserving algorithm, summing over all intervals we obtain that for the given input sequence

$$\omega_{\text{Opt}}(\mathcal{I}) \leq \sum_j \omega(S_j). \quad (1)$$

Consider now the schedule of Greedy. Let S'_j denote the set of $B_{\min}$ most valuable packets absorbed during I_j , let S''_j denote the $B_{\min}$ most valuable packets stored in one of the buffers in the system when the next interval I_{j+1} starts, and let S_j^* denote the $B_{\min}$ most valuable packets from $S'_j \cup S''_j$. Note that S_j^* is exactly the set of $B_{\min}$ most valuable packets that were in the system during I_j and were not dropped. We claim that

$$\omega(S_j^*) \geq \frac{B_{\min}}{2DR} \omega(S_j). \quad (2)$$

To see that, note that a packet $p \in S_j$ is dropped from a buffer Q_e only if Q_e contains at least $\text{size}(Q_e) \geq B_{\min}$ packets with value greater than $\omega(p)$. To complete the proof of the theorem, observe that for all j we have that $\omega(S'_j) \geq \omega(S''_{j-1})$, i.e., the value absorbed in an interval is at least the total value of the $B_{\min}$ most valuable packets stored when the interval starts. Hence, using Eqs. (1,2), and since $S_j^* \subseteq S'_j \cup S''_j$, we get

$$\begin{aligned} \omega_{\text{Opt}}(\mathcal{I}) &\leq \sum_j \omega(S_j) \leq \frac{2DR}{B_{\min}} \sum_j \omega(S_j^*) \\ &\leq \frac{2DR}{B_{\min}} \left(\sum_j \omega(S'_j) + \sum_j \omega(S''_j) \right) \\ &\leq 4 \frac{DR}{B_{\min}} \sum_j \omega(S'_j) = \frac{4DR}{B_{\min}} \cdot \omega_{\text{Greedy}}(\mathcal{I}). \quad \blacksquare \end{aligned}$$

One immediate corollary of Theorem 4 is that the lower bound of Theorem 1 is tight, as implied by the result below.

Corollary 1. *In a tree-topology system where all nodes have identical buffer size and all links have the same bandwidth, the competitive factor of Greedy is $O(\min(h, \alpha))$, where h is the depth of the tree and α is the ratio between the most and the least valuable packets in the input.*

Proof: For the given system, we have that $D = hB_{\min}/R$ since all buffers have size $B_{\min}$ and all links have bandwidth R . Therefore, by Theorem 4, the competitive factor is at most $O(h)$. To see that the competitive factor is at most $O(\alpha)$, observe that Greedy outputs the maximal possible number of packets. ■

4.2 An Upper Bound for Greedy Schedules on Trees

We now prove our main result, which is an upper bound on the competitive ratio of Greedy for tree topologies with arbitrary buffer sizes and link bandwidths. The result holds under the assumption that all packet values are either 1 or $\alpha > 1$.

We introduce the following key concept. Recall that the delay of a link e , denoted $d(e)$, is the size of its buffer divided by its bandwidth, and the delay of a path is the sum of its links' delays.

Definition 3. *Let $e = (v, u)$ be any link in a given tree topology, and suppose that v has children $v_1, \dots, v_k$. The height of e , denoted $h(e)$, is the maximum path delay, over all paths starting at a leaf and ending at u . The weakness of e , denoted $\lambda(e)$, is defined to be $\lambda(e) = \frac{h(e)}{d(e)}$.*

Intuitively, $h(e)$ is just an upper bound on the number of time units that a packet can spend in the system before being sent over e . The significance of the notion of weakness of a link is made explicit in the following theorem.

Theorem 5. *The competitive ratio of Greedy for any given tree topology $G = (V, E)$ and two packet values is $O(\max \{\lambda(e) : e \in E\})$.*

Proof: Fix the input sequence. Consider the schedule produced by Greedy. We construct a set of time intervals called *overload intervals*, where each interval is associated with a link. The construction proceeds from the root link inductively as follows. Consider a link e , and suppose that all overload intervals were already defined for all links e' downstream from e . The set of overload intervals at e is defined as follows. For each time point t^* in which a high-value packet is dropped from Q_e , we define an overload interval $I = [t_s, t_f]$ such that

- (1) $t^* \in I$.
- (2) In each time step $t \in I$, $W(e)$ high value packets are sent over e .
- (3) For any overload interval $I' = [t'_s, t'_f]$ of a downstream link e' , we have that either $t_s > t'_f$ or $t_f < t'_s - d(e, e')$, where $d(e, e')$ is the sum of link delays on the path that starts at the endpoint of e and ends at the endpoint of e' .
- (4) I is maximal.

Note that if a high value packet is dropped from a buffer Q_e by Greedy at time t , then Q_e is full of high value packets at time t , and hence $W(e)$ high value packets will be sent over e in each time step $t, t+1, \dots, t+d(e)$. However, the overload interval containing t may be shorter (possibly empty), due to condition (3).

We now define a couple of notions regarding overload intervals. The *dominance* relation between overload intervals is defined as follows. If for an overload interval $I = [t_s, t_f]$ that occurs at link e there exists an overload interval $I' = [t'_s, t'_f]$ that occurs at a downstream link e' such that $t'_s = t_f + d(e, e') + 1$, we say that I is *dominated* by I' . We also define the notion of *full intervals*: an overload interval I that occurs at link e is said to be *full* if $|I| \geq d(e)$. Note that some non-full intervals may be not dominated.

We now proceed with the proof. For the sake of simplicity, we do not attempt to get the tightest possible constant factors. We partition the set of overload intervals so that in each part there is exactly one full interval, by mapping each overload interval I to a full interval denoted $P(I)$. Given an overload interval I , the mapping is done inductively, by constructing a sequence $I_0, \dots, I_\ell$ of overload intervals such that $I = I_0$, $P(I) = I_\ell$, and only interval I_ℓ is full. Let I be any overload interval, and suppose it occurs at link e . We set $I_0 = I$, and let $e_0 = e$. Suppose that we have defined I_j already. If I_j is full, the sequence is complete. Otherwise, by definition of overload intervals, there must exist another interval I_{j+1} at a link e_{j+1} downstream from e_j that dominates I_j . If there is more than one interval dominating I_j , let I_{j+1} be the one that occurs at the lowest level. Note that the sequence must terminate since for all j , e_{j+1} is strictly downstream from e_j .

Let $\mathcal{F}$ denote the set of all full intervals. Let I be a full interval that occurs at link e . Define the set $\mathcal{P}(I) = \{I' : P(I') = I\}$. This set consists of overload intervals that occur at links in the subtree rooted by e . Define the *coverage* of I , denoted $\mathcal{C}(I)$, to be the following time window:

$$\mathcal{C}(I) = \left[\min \{t : t \in I' \text{ for } I' \in \mathcal{P}(I)\} - h(e) , \max \{t : t \in I' \text{ for } I' \in \mathcal{P}(I)\} + h(e) \right]$$

In words, $\mathcal{C}(I)$ starts $h(e)$ time units before the first interval starts in $\mathcal{P}(I)$, and ends $h(e)$ time units after the last interval ends in $\mathcal{P}(I)$. The key arguments of the proof are stated in the following lemmas.

Lemma 1. *For any full interval I that occurs at any link e , $|\mathcal{C}(I)| < |I| + 4h(e)$.*

Proof: Let I_0 be the interval that starts first in $\mathcal{P}(I)$, and let $I_1, \dots, I_\ell$ be the sequence of intervals in $\mathcal{P}(I)$ such that I_{j+1} dominates I_j for all $0 \leq j < \ell$, and such that $I_\ell = I$. For each j , let $I_j = [t_j, t'_j]$, and suppose that I_j occurs at e_j . Note that I_ℓ is also the interval that ends last in $\mathcal{P}(I)$. Since for all $j < \ell$ we have that I_j is not full, and using the definition of the dominance relation, we have that

$$\begin{aligned} |\mathcal{C}(I)| - 2h(e) &= t'_\ell - t_0 = \sum_{j=0}^{\ell} (t'_j - t_j) + \sum_{j=1}^{\ell} (t_j - t'_{j-1}) \\ &< |I| + \sum_{j=0}^{\ell-1} d(e_j) + \sum_{j=1}^{\ell} d(e_{j-1}, e_j) \leq |I| + 2h(e). \quad \blacksquare \end{aligned}$$

Lemma 2. *For each full interval I that occurs at a link e , the total number of high value packets that are ever sent by Opt from e and were dropped by Greedy during $\mathcal{C}(I)$ is at most $W(e) \cdot (|I| + 6h(e))$.*

Proof: As mentioned above, a packet that is dropped from any buffer upstream from e at time t can never be sent by any schedule outside the time window $[t - h(e), t + h(e)]$. The result therefore follows from Lemma 1. ■

Lemma 3. *For each high-value packet p dropped by Greedy from a link e' at time t , there exists a full overload interval I that occurs in a link downstream from e' (possibly e' itself) such that $t \in \mathcal{C}(I)$.*

Proof: We proceed by the case analysis.

If $t \in I'$ for some full overload interval I' of e' , we are done since $t \in \mathcal{C}(I')$.

If $t \in I'$ for some non-full overload interval of e' dominated by another overload interval I , we have that $t \in \mathcal{C}(P(I))$.

If $t \in I'$ for some non-full overload interval $I' = [t'_s, t'_f]$ of e' that is not dominated by any other overload interval then there exists an overload interval I'' that occurs in a link e'' downstream from e' such that $t'_s = t''_f + 1$ and hence $t \in \mathcal{C}(P(I''))$ because $t''_f + d(e') \geq t'_f$.

If t is not in any overload interval of e' then by the construction for an overload interval I'' that occurs in a link e'' downstream from e' we have that $t''_s - d(e', e'') \leq t \leq t''_f$, which implies that $t \in \mathcal{C}(P(I''))$.

■

Lemma 4. *For each overload interval I , Greedy sends at least $|I| \cdot W(e)$ high value packets from e , and these packets are never dropped.*

Proof: The number of packets sent follows from the fact that when a high-value packet is dropped by Greedy from Q_e , the buffer is full of high value packets. The definition of overload intervals ensures that no high value packet during an overload interval is ever dropped, since if a packet that is sent over e at time t is dropped from a downstream buffer e' at time t' , then $t' \leq t + d(e, e')$. ■

We now conclude the proof of Theorem 5. Consider the set of all packets sent by Opt. Since the total number of packets sent by Greedy in a tree topology is maximal, it is sufficient to consider only the high-value packets. By Lemma 3, it is sufficient to consider only the time intervals $\{\mathcal{C}(I) : I \in \mathcal{F}\}$ since outside these intervals Greedy does as well as Opt. For each $I \in \mathcal{F}$ that occurs at a link e , we have by Lemma 4 that Greedy sends at least $|I| \cdot W(e)$ high value packets, whereas by Lemma 2 Opt sends at most $W(e) \cdot (|I| + 6h(e))$ high value packets. The theorem follows. ■

References

1. W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosén. Dynamic routing on networks with fixed-size buffers. In *Proc. of the 14th ann. ACM-SIAM Symposium on Discrete Algorithms*, pages 771–780, Jan. 2003.

2. W. Aiello, Y. Mansour, S. Rajagopalan, and A. Rosen. Competitive queue policies for differentiated services. In *Proc. IEEE INFOCOM*, 2000.
3. Y. Azar and Y. Richter. Management of multi-queue switches in QoS networks. In *Proc. 33rd ACM STOC*, June 2003. To appear.
4. D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Internet RFC 2475, December 1998.
5. A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *J. ACM*, 48(1):13–38, 2001.
6. D. Clark and J. Wroclawski. An approach to service allocation in the Internet. Internet draft, 1997. Available from `diffserv.lcs.mit.edu`.
7. S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, 1993.
8. E. H. Hahne, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. In *Proc. of the 2001 ACM Symposium on Parallel Algorithms and Architecture*, pages 53–58, 2001.
9. S. Keshav. *An engineering approach to computer networking: ATM networks, the Internet, and the telephone network*. Addison-Wesley Longman Publishing Co., Inc., 1997.
10. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. In *Proc. 33rd ACM STOC*, pages 520–529, July 2001.
11. A. Kesselman and Y. Mansour. Loss-bounded analysis for differentiated services. *Journal of Algorithms*, Vol. 46, Issue 1, pages 79–95, January 2003.
12. A. Kesselman and Y. Mansour. Harmonic buffer management policy for shared memory switches. In *Proc. IEEE INFOCOM*, 2002.
13. M. A. Labrador and S. Banerjee. Packet dropping policies for ATM and IP networks. *IEEE Communications Surveys*, 2(3), 1999.
14. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994.
15. Z. Lotker and B. Patt-Shamir. Nearly optimal FIFO buffer management for DiffServ. In *Proc. 21st Ann. ACM Symp. on Principles of Distributed Computing*, pages 134–143, 2002.
16. Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *J. of Algorithms*, 14:449–465, 1993. A preliminary version appears in the *Proc. of 10th Annual Symp. on Principles of Distributed Computing*, 1991.
17. M. May, J.-C. Bolot, A. Jean-Marie, and C. Diot. Simple performance models of differentiated services for the Internet. In *Proc. IEEE INFOCOM*, 1998.
18. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.
19. The ATM Forum Technical Committee. Traffic management specification version 4.0, Apr. 1996. Available from `www.atmforum.com`.
20. A. Veres and M. Boda. The chaotic nature of TCP congestion control. In *Proc. IEEE INFOCOM*, pages 1715–1723, 2000.

Improved Competitive Guarantees for QoS Buffering

Alex Kesselman¹, Yishay Mansour¹, and Rob van Stee^{2,*}

¹ School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.

{alex,mansour}@cs.tau.ac.il

² Centre for Mathematics and Computer Science (CWI), Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands. Rob.van.Stee@cwi.nl

Abstract. We consider a network providing Differentiated Services (Diffserv) which allow Internet service providers (ISP) to offer different levels of Quality of Service (QoS) to different traffic streams. We study FIFO buffering algorithms, where packets must be transmitted in the order they arrive. The buffer space is limited, and packets are lost if the buffer is full. Each packet has an intrinsic value, and the goal is to maximize the total value of transmitted packets. Our main contribution is an algorithm for arbitrary packet values that for the first time achieves a competitive ratio better than 2, namely $2 - \epsilon$ for a constant $\epsilon > 0$.

1 Introduction

Today's prevalent Internet service model is the best-effort model (also known as the "send and pray" model). This model does not permit users to obtain better service, no matter how critical their requirements are, and no matter how much they may be willing to pay for better service. With the increased use of the Internet for commercial purposes, such a model is not satisfactory any more. However, providing any form of stream differentiation is infeasible in the core of the Internet.

Differentiated Services were proposed as a compromise solution for the Internet Quality of Service (QoS) problem. In this approach each packet is assigned a predetermined QoS, thus aggregating traffic to a small number of classes [3]. Each class is forwarded using the same per-hop behavior at the routers, thereby simplifying the processing and storage requirements. Over the past few years Differentiated Services has attracted a great deal of research interest in the networking community [18,6,16,13,12,5]. We abstract the DiffServ model as follows: packets of different QoS priority have distinct values and the system obtains the value of a packet that reaches its destination.

To improve the network utilization, most Internet Service Providers (ISP) allow some under-provisioning of the network bandwidth employing the policy known as *statistical multiplexing*. While statistical multiplexing tends to be very cost-effective, it requires satisfactory solutions to the unavoidable events of overload. In this paper we study such scenarios in the context of *buffering*. More specifically, we consider an output port of a network switch with the following activities. At each time step, an arbitrary set of

* Work supported by the Deutsche Forschungsgemeinschaft, Project AL 464/3-1, and by the European Community, Projects APPOL and APPOL II. Work partially supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-061-000.

packets arrives, but only one packet can be transmitted. A buffer management algorithm has to serve each packet online, i.e. without knowledge of future arrivals. The algorithm performs two functions: selectively rejects and preempts packets, subject to the buffer capacity constraint, and decides which packet to send. The goal is to maximize the total value of packets transmitted.

In the classical *First-In-First-Out* (FIFO) model packets can not be sent out of order. Formally, for any two packets p, p' sent at times t, t' , respectively, we have that if $t' > t$, then packet p has not arrived after packet p' . If packets arrive at the same time, we refer to the order in which they are processed by the buffer management algorithm, which receives them one by one. Most of today's Internet routers deploy the FIFO buffering policy. Since the buffer size is fixed, when too many packets arrive, *buffer overflow* occurs and some packets must be discarded.

Giving a realistic model for Internet traffic is a major problem in itself. Network arrivals have often been modeled as a Poisson process both for ease of simulation and analytic simplicity and initial works on DiffServ have focused on such simple probabilistic traffic models [11,15]. However, recent examinations of Internet traffic [14,19] have challenged the validity of the Poisson model. Moreover, measurements of real traffic suggest the existence of significant traffic variance (burstiness) over a wide range of time scales.

We analyze the performance of a buffer management algorithm by means of competitive analysis. Competitive analysis, introduced by Sleator and Tarjan [17] (see also [4]), compares an on-line algorithm to an optimal offline algorithm OPT , which knows the entire sequence of packet arrivals in advance. Denote the value earned by an algorithm ALG on an input sequence σ by $V_{\text{ALG}}(\sigma)$.

Definition 1. An online algorithm ALG is c -competitive iff for every input sequence σ , $V_{\text{OPT}}(\sigma) \leq c \cdot V_{\text{ALG}}(\sigma)$.

An advantage of competitive analysis is that a uniform performance guarantee is provided over all input instances, making it a natural choice for Internet traffic.

In [1] different non-preemptive algorithms are studied for the two distinct values model. Recently, this work has been generalized to multiple packet values [2], where they also present a lower bound of $\sqrt{2}$ on the performance of any online algorithm in the preemptive model. Analysis of preemptive queuing algorithms for arbitrary packet values in the context of smoothing video streams appears in [10]. This paper establishes an impossibility result, showing that no online algorithm can have a competitive ratio better than $5/4$, and demonstrates that the greedy algorithm is at least 4-competitive. In [7] the greedy algorithm has been shown to achieve the competitive ratio of 2. The loss of an algorithm is analyzed in [8], where they present an algorithm with competitive ratio better than 2 for the case of two and exponential packet values. In [9] they study the case of two packet values and present a 1.3-competitive algorithm. The problem of whether the competitive ratio of 2 of the natural greedy algorithm can be improved has been open for a long time. In this paper we solve it positively. Our model is identical to that of [7].

Our Results. The main contribution of this paper is an algorithm for the FIFO model for *arbitrary* packet values that achieves a competitive ratio of $2 - \epsilon$ for a constant $\epsilon > 0$.

In particular, this algorithm accomplishes a competitive ratio of 1.983 for a particular setting of parameters. This is the first upper bound below the bound of 2 that was shown in [7]. We also show a lower bound of 1.419 on the performance of any online algorithm, improving on [2], and a specific lower bound of $\phi \approx 1.618$ on the performance of our algorithm.

2 Model Description

We consider a QoS buffering system that is able to hold B packets. The buffer management algorithm has to decide at each step which of the packets to drop and which to transmit, subject to the buffer capacity constraint. The value of packet p is denoted by $v(p)$. The system obtains the value of the packets it sends, and the aim of the buffer management algorithm is to maximize the total value of the transmitted packets. Time is slotted. At the beginning of a time step a set of packets (possibly empty) arrives and at the end of time step a packet is scheduled if any. We denote by $\mathcal{A}(t)$ the set of packets arriving at time step t , by $\mathcal{Q}(t)$ the set of packets in the buffer after the arrival phase at time step t , and by $\text{ALG}(t)$ the packet sent (or scheduled/served) at the end of time step t if any by an algorithm ALG. At any time step t , $|\mathcal{Q}(t)| \leq B$ and $|\text{ALG}(t)| \leq 1$, whereas $|\mathcal{A}(t)|$ can be arbitrarily large. We also denote by $\mathcal{Q}(t, \geq w)$ the subset of $\mathcal{Q}(t)$ of packets with value at least w .

As mentioned in the introduction, we consider FIFO buffers in this paper. Therefore, the packet transmitted at time t is always the first (oldest) packet in the buffer among the packets in $\mathcal{Q}(t)$.

3 Algorithm PG

The main idea of the algorithm PG is to make proactive preemptions of low value packets when high value packets arrive. The algorithm is similar to the one presented in [8], except that each high value packet can preempt at most one low value packet. Intuitively, we try to decrease the delay that a high value packet suffers due to low value packets preceding it in the FIFO order. A formal definition is given in Figure 1.

The parameter of PG is the preemption factor β . For sufficiently large values of β , PG performs like the greedy algorithm and only drops packets in case of overflow. On the other hand, too small values of β can cause excessive preemptions of packets and a large loss of value. Thus, we need to optimize the value of β in order to achieve a balance between maximizing current throughput and minimizing potential future loss.

The following lemma is key to showing a competitive ratio below 2. It shows that if the buffer contains a large number of “valuable” packets then PG sends packets with non-negligible value. This does not hold for the greedy algorithm [7].

Lemma 1. *If at time t , $|\mathcal{Q}(t, \geq w)| \geq B/2$ and the earliest packet from $\mathcal{Q}(t, \geq w)$ arrived before or at time $t - B/2$ then the packet scheduled at the next time step has value at least w/β .*

1. When a packet p of value $v(p)$ arrives, drop the first packet p' in the FIFO order such that $v(p') \leq v(p)/\beta$, if any (p' is preempted).
2. Accept p if there is free space in the buffer.
3. Otherwise, drop (reject) the packet p' that has minimal value among p and the packets in the buffer. If $p' \neq p$, accept p (p pushes out p').

Fig. 1. Algorithm PG.

Proof. Let p be the first packet from $\mathcal{Q}(t, \geq w)$ in the FIFO order and let $t' \leq t - B/2$ be the arrival time of p . Let X be the set of packets with value less than w/β that were in the buffer before p at time t' . We show that no packet from X is present in the buffer at time $t + 1$. We have $|X| \leq B$. At least $B/2$ packets are served between t' and t . All these packets preceded p since p is still in the buffer at time t . So at most $B/2$ packets in X are not (yet) served at time t . However, at least $B/2$ packets with value greater than or equal to w have arrived by time t and each of them preempts from the buffer the first packet in the FIFO order with value of at most w/β , if any. This shows that all packets in X have been either served or dropped by time t . $\square$

In general, we want to assign the value of packets that OPT serves and PG drops to packets served by PG. Note that the schedule of PG contains a sequence of packet rejections and preemptions. We will add structure to this sequence and give a general assignment method based on *overload intervals*.

3.1 Overload Intervals

Before introducing a formal definition, we will give some intuition. Consider a time t at which a packet of value α is rejected and α is the largest value among the packets that are rejected at this time. Note that all packets in the buffer at the end of time step t have value at least α . Such an event defines an α -overloaded interval $\mathcal{I} = [t_s, t_f)$, which starts at time $t_s = t$.

In principle, $\mathcal{I}$ ends at the last time at which a packet in $\mathcal{Q}(t)$ is scheduled (i.e. at time $t + B - 1$ or earlier). However, in case at some time $t' > t$ a packet of value γ is rejected, γ is the largest value among the packets that are rejected at this time, and a packet from $\mathcal{Q}(t)$ is still present in the buffer, we proceed as follows.

If $\gamma = \alpha$, we extend $\mathcal{I}$ to include t' . In case $\gamma > \alpha$, we start a new interval with a higher overload value. Otherwise, if $\gamma < \alpha$, a new interval begins when the first packet from $\mathcal{Q}(t') \setminus \mathcal{Q}(t)$ is eventually scheduled if any. Otherwise, if all packets from $\mathcal{Q}(t') \setminus \mathcal{Q}(t)$ are preempted, we create a zero length interval $\mathcal{I}' = [t_f, t_f)$ whose overload value is γ . Next we define the notion of overload interval more formally.

Definition 2. An α -overflow takes place when a packet of value α is rejected, where α is said to be the overload value.

Definition 3. A packet p is said to be associated with interval $[t, t')$ if p arrived later than the packet scheduled at time $t - 1$ if any and earlier than the packet scheduled at time t' if any.

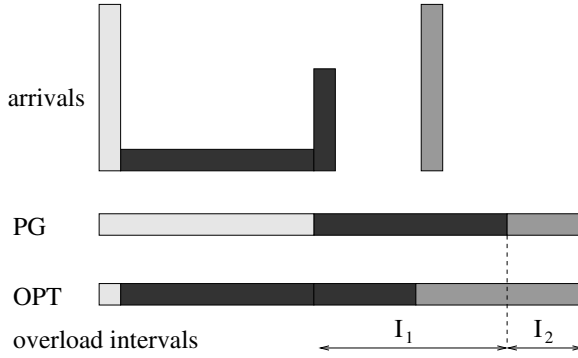


Fig. 2. An example of overload intervals. Light packets have value 1, dark packets value $\beta - \epsilon$, medium packets value 2. The arrival graph should be interpreted as follows: B packets of value 1 arrive at time 1, 1 packet of value $\beta - \epsilon$ arrives at times $2, \dots, B - 1$, etc. Note that I_2 does not start until I_1 is finished.

Intuitively, p is associated with the interval in which it is scheduled, or in which it would have been scheduled if it had not been dropped.

Definition 4. An interval $\mathcal{I} = [t_s, t_f)$, with $t_f \geq t_s$, is an α -overloaded interval if the maximum value of a rejected packet associated with it is α , all packets served during $\mathcal{I}$ were present in the buffer in time of an α -overflow, and $\mathcal{I}$ is a maximal such interval that does not overlap overload intervals with higher overload values.

Thus, we construct overload intervals starting from the highest overload value and ending with the lowest overload value. We note that only packets with value at least α are served during an α -overloaded interval.

Definition 5. A packet p belongs to an α -overloaded interval $\mathcal{I} = [t_s, t_f)$ if p is associated with $\mathcal{I}$ and (i) p is served during $\mathcal{I}$, or (ii) p is rejected no earlier than the first and no later than the last α -overflow, or (iii) p is preempted and it arrived no earlier than the first and no later than the last packet that belongs to $\mathcal{I}$ that is served or rejected.

Whenever an α -overloaded interval $\mathcal{I}$ is immediately followed by a γ -overloaded interval $\mathcal{I}'$ with $\gamma > \alpha$, we have that in the first time step of $\mathcal{I}'$ a packet of value γ is rejected. This does not hold if $\gamma < \alpha$. We give an example in Figure 2.

The following observation states that overload intervals are well-defined.

Observation 1 A rejected packet belongs to exactly one overload interval and overload intervals are disjoint.

Next we introduce some useful definitions related to an overload interval. A packet p *transitively preempts* a packet p' if p either preempts p' or p preempts or pushes out another packet p'' , which transitively preempts p' . A packet p *replaces* a packet p' if (1) p transitively preempts p' and (2) p is eventually scheduled. A packet p *directly replaces* p' if in the set of packets transitively preempted by p no packet except p' is preempted (e.g. p may push out p'' that preempts p').

1. Assign the value of each packet from $\text{PG} \cap \text{OPT}$ to itself.
2. Assign the value of each *preempted* packet from DROP to the packet *replacing* it.
3. Consider all overload sequences starting from the earliest one and up to the latest one. Assign the value of each *rejected* packet from DROP that belongs to the sequence under consideration using the assignment routine for the overload sequence.

Fig. 3. Main assignment routine.

Definition 6. For an overload interval $\mathcal{I}$ let $\text{BELONG}(\mathcal{I})$ denote the set of packets that belong to $\mathcal{I}$. This set consists of three distinct subsets: scheduled packets ($\text{PG}(\mathcal{I})$), preempted packets ($\text{PREEMPT}(\mathcal{I})$) and rejected packets ($\text{REJECT}(\mathcal{I})$). Finally, denote by $\text{REPLACE}(\mathcal{I})$ the set of packets that replace packets from $\text{PREEMPT}(\mathcal{I})$. These packets are either in $\text{PG}(\mathcal{I})$ or are served later.

We divide the schedule of PG into maximal sequences of consecutive overload intervals of increasing and then decreasing overload value.

Definition 7. An overload sequence S is a maximal sequence containing intervals $\mathcal{I}_1 = [t_s^1, t_f^1], \mathcal{I}_2 = [t_s^2, t_f^2], \dots, \mathcal{I}_k = [t_s^k, t_f^k]$ with overload values $w_1, \dots, w_k$ such that $t_f^i = t_s^{i+1}$ for $1 \leq i \leq k-1$, $w_i < w_{i+1}$ for $1 \leq i \leq m-1$ and $w_i > w_{i+1}$ for $m \leq i \leq k-1$, where k is the number of intervals in S and w_m is the maximal overload value among the intervals within S .

Ties are broken by associating an overload interval with the latest overload sequence. We will abbreviate $\text{BELONG}(\mathcal{I}_i), \text{PG}(\mathcal{I}_i), \dots$ by $\text{BELONG}_i, \text{PG}_i, \dots$. We make the following observation, which follows from the definition of an overload interval.

Observation 2 For $1 \leq i \leq k$, all packets in REJECT_i have value at most w_i while all packets in PG_i have value at least w_i .

3.2 Analysis of the PG Algorithm

In the sequel we fix an input sequence σ . Let us denote by OPT and PG the set of packets scheduled by OPT and PG, respectively. We also denote by DROP the set of packets scheduled by OPT and dropped by PG, that is $\text{OPT} \setminus \text{PG}$. In a nutshell, we will construct a *fractional* assignment in which we will assign to packets in PG the value $V_{\text{OPT}}(\sigma)$ so that each packet is assigned at most a $2 - \epsilon$ fraction of its value. The general assignment scheme is presented in Figure 3.

Before we describe the overload sequence assignment routine we need some definitions. Consider an overload sequence S . We introduce the following notation: $\text{OPT}_i = \text{OPT} \cap \text{BELONG}_i$, $\text{REJOPT}_i = \text{OPT} \cap \text{REJECT}_i$, $\text{PRMOPT}_i = \text{OPT} \cap \text{PREEMPT}_i$. We write $\text{PG}(S) = \bigcup_{i=1}^k \text{PG}_i$ and define analogously $\text{OPT}(S)$, $\text{REJOPT}(S)$, and $\text{PRMOPT}(S)$.

Definition 8. For $1 \leq i \leq k$, OUT_i is the set of packets that have been replaced by packets outside S .

Clearly, $\text{OUT}_i \subseteq \text{PREEMPT}_i$. Two intervals $\mathcal{I}_i$ and $\mathcal{I}_j$ are called *adjacent* if either $t_f^i = t_s^j$ or $t_s^i = t_f^j$. The next observation will become important later.

Observation 3 For an interval $\mathcal{I}_i$, if $|\text{PG}_i| + |\text{OUT}_i| < B$ then $\mathcal{I}_i$ is adjacent to another interval $\mathcal{I}_j$ such that $w_j > w_i$.

Suppose that the arrival time of the earliest packet in $\text{BELONG}(S)$ is t_a and let $\text{EARLY}(S) = \bigcup_{t=t_a}^{t_s^1-1} \text{PG}(t)$ be the set of packets sent between t_a and time t_s^1 . Intuitively, packets from $\text{EARLY}(S)$ are packets outside S that interact with packets from S and may be later assigned some value of packets from $\text{DROP}(S)$.

Let $\text{PREVP}(S)$ be the subset of $\mathcal{Q}(t_a) \setminus \text{BELONG}(S)$ containing packets preempted or pushed out by packets from $\text{BELONG}(S)$. The next lemma bounds the difference between the number of packets in $\text{OPT}(S)$ and $\text{PG}(S)$.

Lemma 2. For an overload sequence S the following holds: $|\text{OPT}(S)| - |\text{PG}(S)| \leq B + |\text{OUT}(S)| - |\text{PREVP}(S)|$.

Proof. Let t' be the last time during S at which a packet from $\text{BELONG}(S)$ has been rejected. It must be the case that $t_s^k - t' \geq B - |\text{OUT}(S)|$ since at time t' the buffer was full of packets from $\text{BELONG}(S)$ and any packet outside $\text{BELONG}(S)$ can preempt at most one packet from $\text{BELONG}(S)$. We argue that OPT has scheduled at most $t' + 2B - t_s^1 - |\text{PREVP}(S)|$ packets from $\text{BELONG}(S)$. That is due to the fact that the earliest packet from $\text{BELONG}(S)$ arrived at or after time $t_s^1 - B + |\text{PREVP}(S)|$. On the other hand, PG has scheduled at least $t' + B - t_s^1 - |\text{OUT}(S)|$ packets from $\text{BELONG}(S)$, which yields the lemma. $\square$

Definition 9. A packet is available after executing the first two steps of the main assignment routine if it did not directly replace a packet that OPT serves.

An available packet might still have indirectly replaced a packet served by OPT . However, the fact that it did not directly replace such a packet allows us to upper bound the value assigned to it in the first two steps of the assignment routine. We will use this fact later.

The sequence assignment routine presented in Figure 4 assigns the value of all packets from $\text{REJOPT}(S)$. For the sake of analysis, we make some simplifying assumptions.

1. For any $1 \leq i \leq k$, $|\text{REJOPT}_i| \geq |\text{PG}_i \setminus \text{OPT}_i| + |\text{OUT}_i|$.
2. No packet from $\text{EXTRA}(S)$ belongs to another overload sequence (the set $\text{EXTRA}(S)$ will be defined later).

We show that the assignment routine is feasible under the assumptions (1) and (2). Then we derive an upper bound on the value assigned to any packet in PG . Finally, we demonstrate how to relax these assumptions.

First we will use Lemma 1 to show that for each but the $B/2$ largest packets from $\text{UNASG}(S)$, PG has scheduled some extra packet with value that constitutes at least a $1/\beta$ fraction of its value. The following crucial lemma explicitly constructs the set $\text{EXTRA}(S)$ for the sequence assignment routine. Basically, this set will consist of packets that PG served at times that OPT was serving other (presumably more valuable) packets.

1. For interval $\mathcal{I}_i$ s.t. $1 \leq i \leq k$, assign the value of each of the $|\text{PG}_i \setminus \text{OPT}_i| + |\text{OUT}_i|$ most valuable packets from REJOPT_i to a packet in $(\text{PG}_i \setminus \text{OPT}_i) \cup \text{REPLACE}_i$.
2. Let UNASG_i be the subset of REJOPT_i containing packets that remained unassigned, $\text{UNASG}(S) = \bigcup_{i=1}^k \text{UNASG}_i$, and $\text{SMALL}(S)$ be the subset of $\text{UNASG}(S)$ containing the $\max(|\text{UNASG}(S)| - B/2, 0)$ packets with the lowest value. Find a set $\text{EXTRA}(S)$ of packets from $(\text{PG}(S) \setminus \text{PG}_m) \cup \text{EARLY}(S)$ s.t. $|\text{EXTRA}(S)| = |\text{SMALL}(S)|$ and the value of the l -th largest packet in $\text{EXTRA}(S)$ is at least as large as that of the l -th largest packet in $\text{SMALL}(S)$ divided by β . For each *unavailable* packet in $\text{EXTRA}(S)$, remove from it a $\frac{2}{\beta}$ fraction of its value (this value will be reassigned at the next step).
3. Assign the value of each pair of packets from $\text{SMALL}(S)$ and $\text{UNASG}(S) \setminus \text{SMALL}(S)$ to a pair of *available* packets from $\text{PG}_m \cup \text{REPLACE}_m$ and the packet from $\text{EXTRA}(S)$. Assign to these packets also the value removed from the packet in $\text{EXTRA}(S)$, if any. Do this in such a way that each packet is assigned at most $1 - \epsilon$ times its value.
4. Assign a $1 - 1/\beta$ fraction of the value of each packet from $\text{UNASG}(S)$ that is not yet assigned to an *available* packet in $\text{PG}_m \cup \text{REPLACE}_m$ that has not been assigned any value at Step 3 or the current step of this assignment routine and a $1/\beta$ fraction of its value to some packet from $\text{PG}_m \cup \text{REPLACE}_m$ that has not been assigned any value at Step 3 or the current step of this assignment routine (note that this packet may have been assigned some value by the main routine).

Fig. 4. Overload sequence assignment routine.

Lemma 3. *For an overload sequence S , we can find a set $\text{EXTRA}(S)$ of packets from $(\text{PG}(S) \setminus \text{PG}_m) \cup \text{EARLY}(S)$ such that $|\text{EXTRA}(S)| = |\text{SMALL}(S)|$ and the value of the l -th largest packet in $\text{EXTRA}(S)$ is at least as large as that of the l -th largest packet in $\text{SMALL}(S)$ divided by β .*

Proof. By definition, $|\text{SMALL}(S)| = \max(|\text{UNASG}(S)| - B/2, 0)$. To avoid trivialities, assume that $|\text{UNASG}(S)| > B/2$ and let $x_i = |\text{UNASG}_i|$. By assumption (1)

$$x_i = |\text{REJOPT}_i| - |\text{PG}_i \setminus \text{OPT}_i| - |\text{OUT}_i| \geq 0.$$

Thus

$$\begin{aligned} |\text{OPT}_i \setminus \text{PRMOPT}_i| &= |\text{REJOPT}_i| + |\text{OPT}_i \cap \text{PG}_i| \\ &= x_i + |\text{PG}_i \setminus \text{OPT}_i| + |\text{OPT}_i \cap \text{PG}_i| + |\text{OUT}_i| \\ &= x_i + |\text{PG}_i| + |\text{OUT}_i|. \end{aligned}$$

Let PREDOPT_i be the set of packets from $\text{OPT}_i \setminus \text{PRMOPT}_i$ that have been scheduled by OPT before time t_s^i . We must have $|\text{PREDOPT}_i| \geq x_i$ since the buffer of PG is full of packets from $\bigcup_{j=\min(i,m)}^k \text{BELONG}_j$ at time t_s^j . If it is not the case then we obtain that the schedule of OPT is infeasible using an argument similar to that of Lemma 2.

We also claim that $|\text{PREDOPT}_m| \geq \sum_{i=m}^k x_i$ and PREDOPT_m contains at least $\sum_{i=m+1}^k x_i$ packets with value greater than or equal to w_m . Otherwise the schedule of OPT is either infeasible or can be improved by switching a packet $p \in \bigcup_{i=m+1}^k (\text{OPT}_i \setminus \text{PG}_i)$ and a packet $p' \in \text{BELONG}_m \setminus \text{OPT}_m$ s.t. $v(p) < w_m$ and $v(p') \geq w_m$.

Let MAXUP_j be the set of the x_j most valuable packets from PREDOPT_j for $1 \leq j < m$. It must be the case that the value of the l -th largest packet in MAXUP_j is at least as large

as that of the l -th largest packet in UNASG_j for $1 \leq l \leq |\text{UNASG}_j|$. That is due to the fact that by Observation 2 the x_j least valuable packets from REJOPT_j are also the x_j least valuable packets from $\text{REJOPT}_j \cup \text{PG}_j$.

Now for j starting from k and down to $m - 1$, let MAXDOWN_j be the set containing x_j arbitrary packets from $\text{PREDOPT}_m \setminus (\cup_{i=m+1}^{j-1} \text{MAXDOWN}_i)$ with value at least w_m . (Recall that PREDOPT_m contains at least $\sum_{i=m+1}^k x_i$ packets with value greater than or equal to w_m .) Finally, let MAXUP_m be the set of the x_m most valuable packets from $\text{PREDOPT}_m \setminus (\cup_{i=m+1}^k \text{MAXDOWN}_i)$. Clearly, any packet in MAXDOWN_j has greater value than any packet in REJECT_j for $m + 1 \leq j \leq k$. Similarly to the case of $j < m$, we obtain that the value of the l -th largest packet in MAXUP_m is at least as large as that of the l -th largest packet in UNASG_m for $1 \leq l \leq |\text{UNASG}_m|$.

Let $\text{MAXP}(S) = (\cup_{i=1}^m \text{MAXUP}_i) \cup (\cup_{i=m+1}^k \text{MAXDOWN}_i)$ and let t_i be the time at which OPT schedules the i -th packet from $\text{MAXP}(S)$. We also denote by $\text{MAXP}(S, t_i)$ the set of packets from $\text{MAXP}(S)$ that arrived by time t_i . For $B/2 + 1 \leq i \leq |\text{UNASG}(S)|$, let $\text{LARGE}(t_i)$ be the set of $B/2$ largest packets in $\text{MAXP}(S, t_i)$. We define

$$\text{EXTRA}(S) = \cup_{i=B/2+1}^{|\text{UNASG}(S)|} \text{PG}(t_i).$$

That is, the set $\text{EXTRA}(S)$ consists of the packets served by PG while OPT was serving packets from the PREDOPT sets.

We show that at time t_i , PG schedules a packet with value of at least w'/β , where w' is the minimal value among packets in $\text{LARGE}(t_i)$. If all packets from $\text{LARGE}(t_i)$ are present in the buffer at time t_i then we are done by Lemma 1. Note that the earliest packet from $\text{LARGE}(t_i)$ arrived before or at time $t_i - B/2$ since OPT schedules all of them by time t_i . In case a packet p from $\text{LARGE}(t_i)$ has been dropped, then by the definition of PG and the construction of the intervals, PG schedules at this time a packet that has value at least $v(p) > w'/\beta$.

Observe that the last packet from $\text{EXTRA}(S)$ is sent earlier than t_s^m and therefore $\text{EXTRA}(S) \cap \text{PG}_m = \emptyset$. It is easy to see that the set defined above satisfies the condition of the lemma. $\square$

Theorem 1. *The mapping routine is feasible.*

Proof. If all assignments are done at Step 1 or Step 2 of the main assignment routine then we are done. Consider an overload sequence S that is processed by the sequence assignment routine. By Lemma 2, we obtain that the number of unassigned packets is bounded from above by:

$$\begin{aligned} |\text{UNASG}(S)| &= |\text{REJOPT}(S)| + |\text{PG}(S) \cap \text{OPT}(S)| - |\text{PG}(S)| - |\text{OUT}(S)| \\ &= |\text{OPT}(S)| - |\text{PRMOPT}(S)| - |\text{PG}(S)| - |\text{OUT}(S)| \\ &\leq B - |\text{PRMOPT}(S)| - |\text{PREVP}(S)|. \end{aligned} \tag{1}$$

Observe that each packet p that replaces a packet p' with value w can be assigned a value of w if $p' \in \text{OPT}$. In addition, if p' belongs to another overload sequence S' then p can be assigned an extra value of w at Step 3 or Step 4 of the sequence assignment routine.

Let ASG_1 be the subset of $PG_m \cup REPLACE_m$ containing the unavailable packets after the first two steps of the main assignment routine. By definition, every such packet directly replaced a packet from OPT . We show that all packets directly replaced by packets from ASG_1 belong to $PRMOPT(S) \cup PREV(P(S))$. Consider such a packet p . If p is directly preempted by a packet from ASG_1 then we are done. Else, we have that p is preempted by a packet p' , which is pushed out (directly or indirectly) by a packet from ASG_1 . In this case, by the overload sequence construction, p' must belong to S , and therefore p belongs to $PRMOPT(S) \cup PREV(P(S))$. Thus, $|ASG_1| \leq |PRMOPT(S)| + |PREVP(S)|$.

We denote by ASG_2 the subset of $PG_m \cup REPLACE_m$ containing packets that have been assigned some value at Step 3 of the sequence assignment routine. We have $|ASG_2| = 2 \max(|UNASG(S)| - B/2, 0)$.

Finally, let ASG_3 and ASG_4 be the subsets of $PG_m \cup REPLACE_m$ containing packets that have been assigned at Step 4 of the sequence assignment routine a $1 - 1/\beta$ and a $1/\beta$ fraction of the value of a packet from $UNASG(S)$, respectively. Then $|ASG_3| = |ASG_4| = |UNASG(S)| - 2 \max(|UNASG(S)| - B/2, 0)$.

Now we will show that the assignment is feasible. By (1), we have that

$$|ASG_1| + |ASG_2| + |ASG_3| \leq B$$

while Observation 3 implies that $|PG_m \cup REPLACE_m| \geq B$. Finally,

$$|ASG_4| \leq B - |ASG_2| - |ASG_3|,$$

which follows by case analysis. This implies that during the sequence assignment routine we can always find the packets that we need. $\square$

Theorem 2. *Any packet from PG is assigned at most a $2 - \epsilon(\beta)$ fraction of its value, where $\epsilon(\beta) > 0$ is a constant depending on β .*

For the proof, and the calculation of $\epsilon(\beta)$, we refer to the full paper. Optimizing the value of β , we get that for $\beta = 15$ the competitive ratio of PG is close to 1.983, that is $\epsilon(\beta) \approx 0.017$.

Now let us go back to the assumption (1), that is $x_i = |REJOPT_i| - (|PG_i \setminus OPT_i| + |OUT_i|) \geq 0$. We argue that there exist two indices $l \leq m$ and $r \geq m$ s.t. $x_i \geq 0$ for $l \leq i \leq r$ and $x_i \leq 0$ for $1 \leq i < l$ or $l < i \leq k$. In this case we can restrict our analysis to the subsequence of S containing the intervals $\mathcal{I}_l, \dots, \mathcal{I}_r$.

For a contradiction, assume that there exist two indices i, j s.t. $i < j \leq m$ or $i > j \geq m$, $x_i > 0$ and $x_j < 0$. Then there are a packet $p \in OPT_i$ and a packet $p' \in PG_j \setminus OPT_j$ s.t. $v(p') > v(p)$. We obtain that the schedule of OPT can be improved by switching p and p' .

It remains to consider the assumption (2), that is no packet from $EXTRA(S)$ belongs to another overload sequence S' . In this case we improve the bound of Lemma 2 applied to both sequences.

Lemma 4. *For any two consecutive overload sequences S' and S the following holds: $|OPT(S)| + |OPT(S')| - |PG(S)| - |PG(S')| \leq 2B + |OUT(S)| - |PREVP(S)| - |PREVP(S')| - |EXTRA(S) \cap BELONG(S')|$.*

Proof. According to the proof of Lemma 2, $t_f^m - t_l \geq B - |\text{OUT}(S)|$ where t_l is the last time during S at which a packet from $\text{BELONG}(S)$ has been rejected. Let $z = |\text{EXTRA}(S) \cap \text{BELONG}(S')|$. We argue that OPT has scheduled at most $t_l + 2B - t_s^1 - |\text{PREVP}(S')|$ packets from $\text{BELONG}(S) \cup \text{BELONG}(S')$. That is due to the fact that the earliest packet from $\text{BELONG}(S')$ arrived at or after time $t_s^1 - B + |\text{PREVP}(S')|$. Observe that between time t_s^1 and time t_f^k at most $B - z - |\text{PREVP}(S)|$ packets outside of $\text{BELONG}(S) \cup \text{BELONG}(S')$ have been scheduled by PG. Hence, PG has scheduled at least $t_l + z + |\text{PREVP}(S)| - t_s^1 - |\text{OUT}(S)|$ packets from $\text{BELONG}(S) \cup \text{BELONG}(S')$, which yields the lemma. $\square$

Using Lemma 4, we can extend our analysis to any number of consecutive overload sequences without affecting the resulting ratio.

3.3 Lower Bounds

Theorem 3. *The PG algorithm has a competitive ratio of at least ϕ .*

We omit the proof due to space constraints.

Define $v^* = \sqrt[3]{19 + 3\sqrt{33}}$ and $\mathcal{R} = (19 - 3\sqrt{33})(v^*)^2/96 + v^*/6 + 2/3 \approx 1.419$.

Theorem 4. *Any online algorithm ALG has a competitive ratio of at least $\mathcal{R}$.*

Proof. Suppose that ALG maintains a competitive ratio less than $\mathcal{R}$ and let $v = v^*/3 + 4/(3v^*) + 4/3 \approx 2.839$. We define a sequence of packets as follows. At time $t = 1$, B packets with value 1 arrive. At each time $2, \dots, l_1$, a packet of value v arrives, where $t + l_1$ is the time at which ALG serves the first packet of value v (i.e. the time at which there remain no packets of value 1). Depending on l_1 , the sequence either stops at this point or continues with a new phase.

Basically, at the start of phase i , B packets of value v^{i-1} arrive. During the phase, one packet of value v^i arrives at each time step until ALG serves one of them. This is the end of the phase. If the sequence continues until phase n , then in phase n only B packets of value v^{n-1} arrive. Let us denote the length of phase i by l_i for $i = 1, \dots, n-1$ and define $s_i = \sum_{j=1}^i (l_j v^{j-1})/B$ for $i = 1, \dots, n$.

If the sequence stops during phase $i < n$, then ALG earns $l_1 + l_2 v + l_3 v^2 + \dots + l_i v^{i-1} + l_i v^i = B \cdot s_i + l_i v^i$ while OPT can earn at least $l_1 v + l_2 v^2 + \dots + (l_{i-1} + 1) v^{i-1} + l_i v^i = B(v \cdot s_i + v^{i-1})$. The implied competitive ratio is $(v \cdot s_i + v^{i-1})/(s_i + l_i v^i/B)$. We only stop the sequence in this phase if this ratio is at least $\mathcal{R}$, which depends on l_i . We now determine the value of l_i for which the ratio is exactly $\mathcal{R}$. Note that $l_i v^i = (s_i - s_{i-1})/v$. We find

$$\mathcal{R} = \frac{v \cdot s_i + v^{i-1}}{s_i + l_i v^i/B} \Rightarrow s_i = \frac{v \mathcal{R} s_{i-1} + v^{i-1}}{\mathcal{R}(v+1) - v}, s_0 = 0 \Rightarrow s_i = \frac{v^i - (\frac{\mathcal{R}v}{\mathcal{R}(v+1)-v})^i}{(\mathcal{R}-1)v^2}.$$

It can be seen that $s_i/v^i \rightarrow 1/(v^2(\mathcal{R}-1))$ for $i \rightarrow \infty$, since $\mathcal{R}/(\mathcal{R}(v+1)-v) < 1$ for $\mathcal{R} > 1$.

Thus if under ALG the length of phase i is less than l_i , the sequence stops and the ratio is proved. Otherwise, if ALG continues until phase n , it earns $l_1 + l_2 v + l_3 v^2 + \dots + l_n v^{n-1} +$

$B \cdot v^n = B \cdot (s_n + v^n)$ whereas OPT can earn at least $l_1 v + l_2 v^2 + \dots + l_n v^n + B \cdot v^n = B(v \cdot s_n + v^n)$. The implied ratio is

$$\frac{v s_n + v^n}{s_n + v^n} = \frac{v \frac{s_n}{v^n} + 1}{\frac{s_n}{v^n} + 1} \rightarrow \frac{\frac{v}{v^2(\mathcal{R}-1)} + 1}{\frac{1}{v^2(\mathcal{R}-1)} + 1} = \frac{v + v^2(\mathcal{R} - 1)}{1 + v^2(\mathcal{R} - 1)} = \mathcal{R}.$$

□

References

1. W. A. Aiello, Y. Mansour, S. Rajagopalan and A. Rosén, "Competitive Queue Policies for Differentiated Services," *Proceedings of INFOCOM 2000*, pp. 431–440.
2. N. Andelman, Y. Mansour and An Zhu, "Competitive Queueing Policies for QoS Switches," *The 14th ACM-SIAM SODA*, Jan. 2003.
3. Y. Bernet, A. Smith, S. Blake and D. Grossman, "A Conceptual Model for Diffserv Routers," *Internet draft*, July 1999.
4. A. Borodin and R. El-Yaniv, "Online Computation and Competitive Analysis," *Cambridge University Press*, 1998.
5. D. Clark and J. Wroclawski, "An Approach to Service Allocation in the Internet," *Internet draft*, July 1997.
6. C. Dovrolis, D. Stiliadis and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling", *Proceedings of ACM SIGCOMM'99*, pp. 109–120.
7. A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber and M. Sviridenko, "Buffer Overflow Management in QoS Switches," *Proceedings of STOC 2001*, pp. 520–529.
8. A. Kesselman and Y. Mansour, "Loss-Bounded Analysis for Differentiated Services," *Journal of Algorithms*, Vol. 46, Issue 1, pp 79–95, January 2003.
9. Z. Lotker and B. Patt-Shamir, "Nearly optimal FIFO buffer management for DiffServ," *Proceedings of PODC 2002*, pp. 134–142.
10. Y. Mansour, B. Patt-Shamir and Ofer Lapid, "Optimal Smoothing Schedules for Real-Time Streams," *Proceedings of PODC 2000*, pp. 21–29.
11. M. May, J. Bolot, A. Jean-Marie, and C. Diot, "Simple Performance Models of Differentiated Services Schemes for the Internet," *Proceedings of IEEE INFOCOM 1999*, pp. 1385–1394, March 1999.
12. K. Nichols, V. Jacobson and L. Zhang, "A Two-bit Differentiated Services Architecture for the Internet," *Internet draft*, July 1999.
13. T. Nandagopal, N. Venkitaraman, R. Sivakumar and V. Bharghavan, "Relative Delay Differentiation and Delay Class Adaptation in Core-Stateless Networks," *Proceedings of IEEE Infocom 2000*, pp. 421–430, March 2000.
14. V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, Vol. 3, No. 3, pp. 226–244, June 1995.
15. S. Sahu, D. Towsley and J. Kurose, "A Quantitative Study of Differentiated Services for the Internet," *Proceedings of IEEE Global Internet'99*, pp. 1808–1817, December 1999.
16. N. Semret, R. Liao, A. Campbell and A. Lazar, "Peering and Provisioning of Differentiated Internet Services," *Proceedings of INFOCOM 2000*, pp. 414–420, March 2000.
17. D. Sleator and R. Tarjan, "Amortized Efficiency of List Update and Paging Rules," *CACM* 28, pp. 202–208, 1985.
18. I. Stoica and H. Zhang, "Providing Guaranteed Services without Per Flow Management," *Proceedings of SIGCOM 1999*, pp. 81–94.
19. A. Veres and M. Boda, "The Chaotic Nature of TCP Congestion Control," *Proceedings of INFOCOM 2000*, pp. 1715–1723, March 2000.

On Generalized Gossiping and Broadcasting^{*}

(Extended Abstract)

Samir Khuller, Yoo-Ah Kim, and Yung-Chun (Justin) Wan

Department of Computer Science and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742.
{samir,ykim,ycwan}@cs.umd.edu

Abstract. The problems of gossiping and broadcasting have been widely studied. The basic gossip problem is defined as follows: there are n individuals, with each individual having an item of gossip. The goal is to communicate each item of gossip to every other individual. Communication typically proceeds in rounds, with the objective of minimizing the number of rounds. One popular model, called the telephone call model, allows for communication to take place on any chosen matching between the individuals in each round. Each individual may send (receive) a single item of gossip in a round to (from) another individual. In the broadcasting problem, one individual wishes to broadcast an item of gossip to everyone else. In this paper, we study generalizations of gossiping and broadcasting. The basic extensions are: (a) each item of gossip needs to be broadcast to a specified subset of individuals and (b) several items of gossip may be known to a single individual. We study several problems in this framework that generalize gossiping and broadcasting. Our study of these generalizations was motivated by the problem of managing data on storage devices, typically a set of parallel disks. For initial data distribution, or for creating an initial data layout we may need to distribute data from a single server or from a collection of sources.

1 Introduction

The problems of Gossiping and Broadcasting have been the subject of extensive study [21,15,17,3,4,18]. These play an important role in the design of communication protocols in various kinds of networks. The *gossip problem* is defined as follows: there are n individuals. Each individual has an item of gossip that they wish to communicate to everyone else. Communication is typically done in rounds, where in each round an individual may communicate with at most one other individual (also called the telephone model). There are different models that allow for the full exchange of all items of gossip known to each individual in a single round, or allow the sending of only one item of gossip from one to

^{*} Full paper is available at

<http://www.cs.umd.edu/projects/smart/papers/multicast.pdf>. This research was supported by NSF Awards CCR-9820965 and CCR-0113192.

the other (half-duplex) or allow each individual to send an item to the individual they are communicating with in this round (full-duplex). In addition, there may be a communication graph whose edges indicate which pairs of individuals are allowed to communicate in each round. (In the classic gossip problem, communication may take place between any pair of individuals; in other words, the communication graph is the complete graph.) In the *broadcast problem*, one individual needs to convey an item of gossip to every other individual. The two parameters typically used to evaluate the algorithms for this problem are: the number of communication rounds, and the total number of telephone calls placed.

The problems we study are generalizations of the above mentioned gossiping and broadcasting problems. The basic generalizations we are interested in are of two kinds (a) each item of gossip needs to be communicated to only a subset of individuals, and (b) several items of gossip may be known to one individual. Similar generalizations have been considered before [23,25]. (In Section 1.2 we discuss in more detail the relationships between our problem and the ones considered in those papers.)

There are four basic problems that we are interested in. Before we define the problems formally, we discuss their applications to the problem of creating data layouts in parallel disk systems. The communication model we use is the half-duplex telephone model, where only one item of gossip may be communicated between two communicating individuals during a single round. Each individual may communicate (either send or receive an item of data) with at most one other individual in a round. This model best captures the connection of parallel storage devices that are connected on a network and is most appropriate for our application.

We now briefly discuss applications for these problems, as well as prior related work on data migration. To deal with high demand, data is usually stored on a parallel disk system. Data objects are often replicated within the disk system, both for fault tolerance as well as to cope with demand for popular data [29, 5]. Disks typically have constraints on storage as well as the number of clients that can simultaneously access data from it. Approximation algorithms have been developed [26,27,12,19] to map known demand for data to a specific data layout pattern to maximize utilization¹. In the layout, we not only compute how many copies of each item we need, but also a layout pattern that *specifies the precise subset of items on each disk*. The problem is *NP*-hard, but there is a polynomial time approximation scheme [12]. Hence given the relative demand for data, the algorithm computes an almost optimal layout. For example, we may wish to create this layout by copying data from a single source that has all the data initially. Or the data may be stored at different locations initially—these considerations lead to the different problems that we consider.

In our situation, each individual models a **disk** in the system. Each item of gossip is a **data item** that needs to be transferred to a set of disks. If each disk

¹ Utilization refers to the total number of clients that can be assigned to a disk that contains the data they want.

had exactly one data item, and needs to copy this data item to every other disk, then it is exactly the problem of gossiping.

Different communication models can be considered based on how the disks are connected. We use the same model as in the work by [13,1] where the disks may communicate on any matching; in other words, the underlying communication graph is complete. For example, *Storage Area Networks* support a communication pattern that allows for devices to communicate on a specified matching.

Suppose we have N disks and Δ data items. The problems we are interested in are:

1. **Single-source broadcast.** There are Δ data items stored on a single disk (the source). We need to broadcast all items to all $N - 1$ remaining disks.
2. **Single-source multicast.** There are Δ data items stored on a single disk (the source). We need to send data item i to a specified subset D_i of disks. Figure 1 gives an example when Δ is 4.
3. **Multi-source broadcast.** There are Δ data items, each stored separately at a single disk. These need to be broadcast to all disks. We assume that data item i is stored on disk i , for $i = 1 \dots \Delta$.
4. **Multi-source multicast.** There are Δ data items, each stored separately at a single disk. Data item i needs to be sent to a specified subset D_i of disks. We assume that data item i is stored on disk i , for $i = 1 \dots \Delta$.

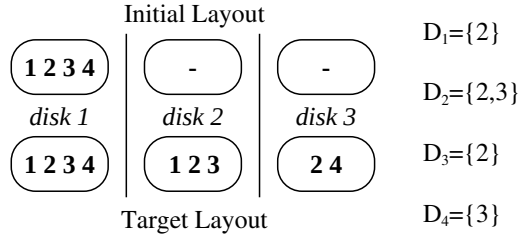


Fig. 1. An initial and target layouts, and their corresponding D_i 's of a single-source multicast instance.

We do not discuss the first problem in any detail since this was solved by [8, 10]. For the multi-source problems, there is a sub-case of interest, namely when the source disks are not in any subset D_i . For this case we can develop better bounds (details omitted).

1.1 Contributions

In Section 2 we define the basic model of communication and the notation used in the paper. Let N be the number of disks and Δ be the number of items. The main results that we show in this paper are:

Theorem 1.1. *For the single-source multicast problem we design a polynomial time algorithm that outputs a solution where the number of rounds is at most $OPT + \Delta$.*

Theorem 1.2. *For the multi-source broadcast problem we design a polynomial time algorithm that outputs a solution where the number of rounds is at most $OPT + 3$.*

Theorem 1.3. *For the multi-source multicast problem we design a polynomial time algorithm that outputs a solution where the number of rounds is at most $4OPT + 3$. We also show that this problem is NP -hard.*

For all the above algorithms, we move data only to disks that need the data. Thus we use no bypass (intermediate) nodes as holding points for the data. If bypass nodes are allowed, we have this result:

Theorem 1.4. *For the multi-source multicast problem allowing bypass nodes we design a polynomial time algorithm that outputs a solution where the number of rounds is at most $3OPT + 6$.*

1.2 Related Work

One general problem of interest is the **data migration problem** when data item i resides in a specified (source) subset S_i of disks, and needs to be moved to a (destination) subset D_i . This problem is more general than the Multi-Source multicast problem where we assumed that $|S_i| = 1$ and that all the S_i 's are disjoint. For the data migration problem we have developed a 9.5-approximation algorithm [20]. While this problem is a generalization of all the problems we study in this paper (and clearly also NP -hard since even the special case of multi-source multicast is NP -hard), the bounds in [20] are not as good. The methods used for single-source multicast and multi-source broadcast are completely different from the algorithm in [20]. Using the methods in [20] one cannot obtain additive bounds from the optimal solution. The algorithm for multi-source multicast presented here is a simplification of the algorithm developed in [20], and we also obtain a much better approximation factor of 4.

Many generalizations of gossiping and broadcasting have been studied before. For example, the paper by Liben-Nowell [23] considers a problem very similar to multi-source multicast with $\Delta = N$. However, the model that he uses is different than the one that we use. In his model, in each telephone call, a pair of users can exchange all the items of gossip that they know. The objective is to simply minimize the total number of phone calls required to convey item i of gossip to set D_i of users. In our case, since each item of gossip is a data item that might take considerable time to transfer between two disks, we cannot assume that an arbitrary number of data items can be exchanged in a single round. Several other papers use the same telephone call model [2,7,14,18,30]. Liben-Nowell [23] gives an exponential time exact algorithm for the problem.

Other related problems that have been studied are the set-to-set gossiping problem [22,25] where we are given two possibly intersecting sets A and B of gossipers and the goal is to minimize the number of calls required to inform all gossipers in A of all the gossip known to members in B . The work by [22] considers minimizing both the number of rounds as well as the total number of

calls placed. The main difference is that in a single round, an arbitrary number of items may be exchanged. For a complete communication graph they provide an exact algorithm for the minimum number of calls required. For a tree communication graph they minimize the number of calls or number of rounds required. Liben-Nowell [23] generalizes this work by defining for each gossip i the set of relevant gossip that they need to learn. This is just like our multi-source multicast problem with $\Delta = N$, except that the communication model is different, as well as the objective function. The work by [9] also studies a set to set broadcast type problem, but the cost is measured as the total cost of the broadcast trees (each edge has a cost). The goal is not to minimize the number of rounds, but the total cost of the broadcast trees. In [11] they also define a problem called scattering which involves one node broadcasting distinct messages to all the other nodes (very much like our single source multicast, where the mutlicast groups all have size one and are disjoint).

As mentioned earlier, the single source broadcast problem using the same communication model as in our paper was solved by [8,10].

2 Models and Definitions

We have N disks and Δ data items. Note that after a disk receives item i , it can be a source of item i for other disks that have not received the item as yet. Our goal is to find a schedule using the minimum number of rounds, that is, to minimize the total amount of time to finish the schedule. We assume that the underlying network is connected and the data items are all the same size, in other words, it takes the same amount of time to migrate an item from one disk to another. The crucial constraint is that each disk can participate in the transfer of only one item—either as a sender or receiver. Moreover, as we do not use any bypass nodes, all data is only sent to disks that desire it.

Our algorithms make use of a known result on edge coloring of multi-graphs. Given a graph G with max degree Δ_G and multiplicity μ the following result is known (see [6] for example). Let χ' be the edge chromatic number of G .

Theorem 2.1. (*Vizing [31]*) *If G has no self-loops then $\chi' \leq \Delta_G + \mu$.*

3 Single-Source Multicasting

In this section, we consider the case where there is one source disk s that has all Δ items and others do not have any item in the beginning. For the case of *broadcasting* all items, it is known that there is a schedule which needs $2\Delta - 1 + \lfloor \log N \rfloor$ rounds for odd N and $\lceil \frac{\Delta(N-1) - 2^{\lfloor \log_2 N \rfloor} + 1}{\lfloor N/2 \rfloor} \rceil + \lfloor \log N \rfloor$ rounds for even N [8,10] and this is optimal. We develop an algorithm that can be applied when D_i is an arbitrary subset of disks. The number of rounds required by our algorithm is at most $\Delta + OPT$ where OPT is the minimum number of rounds required for this problem. Our algorithm is obviously a 2-approximation for the problem, since Δ is a lower bound on the number of rounds required by the optimal solution.

3.1 Outline of the Algorithm

Without loss of generality, we assume that $|D_1| \geq |D_2| \geq \dots \geq |D_\Delta|$ (otherwise renumber the items). Let $|D_i| = 2^{d_i^1} + 2^{d_i^2} + \dots + 2^{d_i^{m_i}}$ where $d_i^j (j = 1, 2, \dots, m_i)$ are integers and $d_i^j > d_i^{j+1}$. (In other words, we consider the bit representation of each $|D_i|$ value.)

Our algorithm consists of two phases.

Phase I. In the first phase, we want to make exactly $\lfloor |D_i|/2 \rfloor$ copies for all items i . At the t -th round, we do the following:

1. If $t \leq \Delta$, copy item t from source s to a disk in D_t .
2. For items j ($j < t$), double the number of copies unless the number of copies reaches $\lfloor |D_j|/2 \rfloor$. In other words, every disk having an item j makes another copy of it if the number of copies of item j is no greater than $2^{d_j^1-2}$, and when it becomes $2^{d_j^1-1}$, then only $\lfloor |D_j|/2 \rfloor - 2^{d_j^1-1}$ disks make copies, and thus the number of copies of item i becomes $\lfloor |D_i|/2 \rfloor$.

Phase II. At t -th round, we finish the migration of item t . Each item j has $\lfloor |D_j|/2 \rfloor$ copies. We finish migrating item t by copying from the current copies to the remaining $\lfloor |D_t|/2 \rfloor$ disks in D_t which did not receive item t as yet, and we use the source disk if $|D_t|$ is odd.

Figure 2 shows an example of data transfers taken in Phase 1.

where $|D_1|$, $|D_2|$ and $|D_3|$ are 8, 6 and 4, respectively. It is easy to see that Phase II can be scheduled without conflicts because we deal with only one item each round. But in Phase I, migration of several items happen at the same time and D_i 's can overlap. Therefore, we may not be able to satisfy the requirement of each round if we arbitrarily choose the disks to receive items. We show that we can finish Phase I successfully without conflicts by carefully choosing disks.

3.2 Details of Phase I

Let D_i^p be the disks in D_i that participate in either sending or receiving item i at the $(i+p)$ -th round. D_i^0 is the first disk receiving i from the source s and

$$|D_i^p| = \begin{cases} 2^p & \text{if } p \leq d_i^1 - 1 \\ 2^{\lfloor \frac{|D_i|}{2} \rfloor} - 2^{d_i^1} & \text{if } p = d_i^1 \end{cases}$$

At $(i+p)$ -th round, disks in $D_j^{i+p-j} (i+p-d_j^1 \leq j \leq \min(i+p, \Delta))$ either send or receive item j at the same time. To avoid conflicts, we decide which disks belong to D_i^p before starting migration. If we choose disks from $D_i \cap D_j$ for D_i^p ($j > i$), it may interfere with the migration of D_j . Therefore, when we build D_i^p , we consider $D_j^{p'}$ where $j > i$ and $p' \leq p$. Also note that since each disk receiving an item should have its corresponding sender, the half of D_i^p should have item i as senders and another half should not have item i as receivers.

We build D_Δ^p first. Choose $2\lfloor |D_\Delta|/2 \rfloor - 2^{d_\Delta^1}$ disks for $D_\Delta^{d_\Delta^1}$ and $2^{d_\Delta^1-1}$ disks for $D_\Delta^{d_\Delta^1-1}$ from D_Δ . When we choose $D_\Delta^{d_\Delta^1-1}$, we should include the half of $D_\Delta^{d_\Delta^1}$

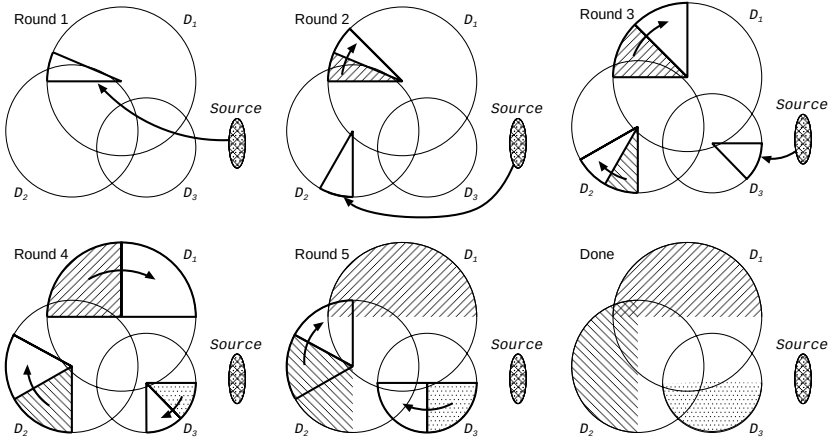


Fig. 2. An example of Phase I when all $|D_i|$ are even

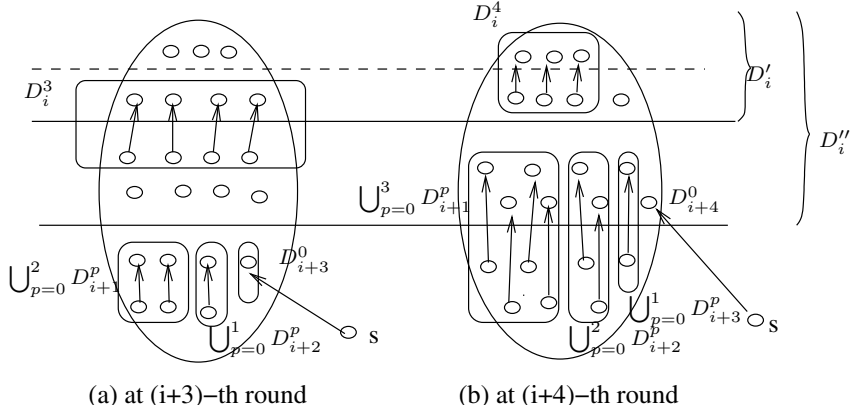


Fig. 3. How disks in D_i behave in Phase I where $|D_i| = 2^4 + 2^2 + 2^1$

(that will be senders at $(\Delta + d_\Delta^1)$ -th round) and exclude the remaining half of $D_\Delta^{d_\Delta^1}$ (that will be receivers at $(\Delta + d_\Delta^1)$ -th round). And then build D_Δ^p ($p < d_\Delta^1 - 1$) by taking any subset of D_Δ^{p+1} .

Now given $D_j^{p'}$ ($i < j \leq \Delta$), we decide D_i^p as follows: Define D'_i to be disks in D_i which do not have any item j ($j > i$) after $(i + d_i^1)$ -th round. In the same way, define D''_i to be disks in D_i which do not have any item j ($j > i$) after $(i + d_i^1 - 1)$ -th round. Formally, since all disks in $\bigcup_{p=0}^{p'} D_j^p$ have item j after $(j + p')$ -th rounds, $D'_i = D_i - \bigcup_{j=i+1}^{\Delta} (\bigcup_{p=0}^{i+d_i^1-j} D_j^p)$ and $D''_i = D_i - \bigcup_{j=i+1}^{\Delta} (\bigcup_{p=0}^{i+d_i^1-1-j} D_j^p)$. As shown in Figure 3, we choose $D_i^{d_i^1}$ from D'_i and also $D_i^{d_i^1-1}$ from D''_i , by which we can avoid conflicts. Also, half of $D_i^{d_i^1}$ should be included in $D_i^{d_i^1-1}$.

(to be senders) and the remaining half should be excluded from $D_i^{d_i^1-1}$ (to be receivers). We make D_i^p ($p < d_i^1 - 1$) by choosing any subset of disks from D_i^{p+1} .

Lemma 3.1. *We can find a migration schedule by which we perform every round in phase I without conflicts.*

3.3 Analysis

We prove that our algorithm uses at most Δ more rounds than the optimal solution for single-source multicasting. Let us denote the optimal makespan of an migration instance I as $C(I)$.

Lemma 3.2. *For any migration instance I , $C(I) \geq \max_{1 \leq i \leq \Delta} (i + \lfloor \log |D_i| \rfloor)$.*

Lemma 3.3. *The total makespan of our algorithm is at most $\max_{1 \leq i \leq \Delta} (i + \lfloor \log |D_i| \rfloor) + \Delta$.*

Theorem 3.4. *The total makespan of our algorithm is at most the optimal makespan plus Δ .*

Corollary 3.5. *We have a 2-approximation algorithm for the single-source multicasting problem.*

4 Multi-source Broadcasting

We assume that we have N disks. Disk i , $1 \leq i \leq \Delta$, has an item numbered i . The goal is to send each item i to all N disks, for all i . We present an algorithm which performs no more than 3 extra rounds than the optimal solution.

4.1 Algorithm Multi-source Broadcast

1. We divide N disks into Δ disjoint sets G_i such that disk $i \in G_i$, for all $i = 1 \dots \Delta$. Let q be $\lfloor \frac{N}{\Delta} \rfloor$ and r be $N - q\Delta$. $|G_i| = q + 1$ for $i = 1 \dots r$, and $|G_i| = q$ for $i = r+1 \dots \Delta$. Every disk in G_i can receive item i using $\lceil \log |G_i| \rceil$ rounds by doubling the item in each round. Since the sets G_i are disjoint, every disk can receive an item belongs to its group in $\lceil \log \frac{N}{\Delta} \rceil$ rounds.
2. We divide all N disks into $q - 1$ groups of size Δ by picking one disk from each G_i , and one group of size $\Delta + r$ which consists of all remaining disks.
3. Consider the first $q - 1$ gossiping groups; each group consists of Δ disks, with each having a distinct item. Using gossiping algorithm in [4], every disk in the first $q - 1$ groups can receive all Δ items in 2Δ rounds².
4. Consider the last gossiping group, there are exactly two disks having items $1, \dots, r$, while there is exactly one disk having item $r + 1, \dots, \Delta$. If r is zero, we can finishes all transfers in 2Δ rounds using algorithm in [4]. For non-zero r , we claim that all disks in this gossiping group can receive all items in 2Δ rounds.

² The number of rounds required is 2Δ if Δ is odd, otherwise it is $2(\Delta - 1)$

We divide the disks in this gossiping group into 2 groups, G_X and G_Y of size $\Delta - \lfloor \frac{\Delta-r}{2} \rfloor$ and $r + \lfloor \frac{\Delta-r}{2} \rfloor$ respectively. Note that $|G_Y| + 1 \geq |G_X| \geq |G_Y|$. Exactly one disk having items $1, \dots, r$ appear in each group, disks having item $r + 1, \dots, \Delta - \lfloor \frac{\Delta-r}{2} \rfloor$ appear in G_X , and the remaining disks (having items $\Delta - \lfloor \frac{\Delta-r}{2} \rfloor + 1, \dots, \Delta$) appear in G_Y . Note that the size of the two groups differ by at most 1. The general idea of the algorithm is as follows (The details of these step are non-trivial and covered in the proof of Lemma 4.1):

- a) Algorithm in [4] is applied to each group in parallel. After this step, each disk has all items belong to its group.
- b) In each round, disks in G_Y send item i to disks in G_X , where i is $\Delta - \lfloor \frac{\Delta-r}{2} \rfloor + 1, \dots, \Delta$. Note that only disks in G_Y have these items, but not the disks in G_X . Since the group sizes diff by at most 1, the number of rounds required is about the same as the number of items transferred.
- c) The step is similar to the above step but in different direction. Item i , where i is $r + 1, \dots, \Delta - \lfloor \frac{\Delta-r}{2} \rfloor$, are copied to G_Y .

Thus, our algorithm takes $\lceil \log \frac{N}{\Delta} \rceil + 2\Delta$ rounds.

4.2 Analysis

Lemma 4.1. *For a group of disks of size $\Delta + r$, where $1 \leq r < \Delta$, if every disk has one item, exactly 2 disks have item $1, \dots, r$, and exactly 1 disk has item $r + 1, \dots, \Delta$, all disks can receive all Δ items in 2Δ rounds.*

Theorem 4.2. *The makespan time of any migration instance of multi-source broadcasting is at least $\lceil \log \frac{N}{\Delta} \rceil + 2(\Delta - 1)$.*

Thus, our solution takes no more than 3 rounds than the optimal.

5 Multi-source Multicasting

We assume that we have N disks. Disk i , $1 \leq i \leq \Delta \leq N$, has data item i . The goal is to copy item i to a subset D_i of disks that do not have item i . (Hence $i \notin D_i$). We could show that finding a schedule with the minimum number of rounds is NP -hard. In this section we present a polynomial time approximation algorithm for this problem. The approximation factor of this algorithm is 4.

We first define β as $\max_{i=1 \dots N} |\{j | i \in D_j\}|$. In other words, β is an upper bound on the number of different sets D_j , that a disk i may belong to. Note that β is a lower bound on the optimal number of rounds, since the disk that attains the max, needs at least β rounds to receive all the items j such that $i \in D_j$, since it can receive at most one item in each round.

The algorithm will first create a small number of copies of each data item j (the exact number of copies will be dependent on $|D_j|$). We then assign each newly created copy to a set of disks in D_j , such that it will be responsible for providing item j to those disks. This will be used to construct a transfer graph,

where each directed edge labeled j from v to w indicates that disk v must send item j to disk w . We will then use an edge-coloring of this graph to obtain a valid schedule [6]. The main difficulty here is that a disk containing an item is its source, is also the destination for several other data items.

Algorithm Multi-source Multicast

1. We first compute a disjoint collection of subsets $G_i, i = 1 \dots \Delta$. Moreover, $G_i \subseteq D_i$ and $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$. (In Lemma 5.1, we will show how such G_i 's can be obtained.)
2. Since the G_i 's are disjoint, we have the source for item i (namely disk i) send the data to the set G_i using $\lceil \log |D_i| \rceil + 1$ rounds as shown in Lemma 5.2. Note that disk i may itself belong to some set G_j . Let $G'_i = \{i\} \cup G_i$. In other words, G'_i is the set of disks that have item i at the end of this step.
3. We now create a transfer graph as follows. Each disk is a node in the graph. We add directed edges from each disk in G'_i to disks in $D_i \setminus G_i$ such that the out-degree of each node in G'_i is at most $\beta - 1$ and the in-degree of each node in $D_i \setminus G_i$ is 1. (In Lemma 5.3 we show how that this can be done.) This ensures that each disk in D_i receives item i , and that each disk in G'_i does not send out item i to more than $\beta - 1$ disks.
4. We now find an edge coloring of the transfer graph (which is actually a multigraph) and the number of colors used is an upper bound on the number of rounds required to ensure that each disk in D_j gets item j . (In Lemma 5.4 we derive an upper bound on the degree of each vertex in this graph.)

Lemma 5.1. [20] *(Step 1) There is a way to choose disjoint sets G_i for each $i = 1 \dots \Delta$, such that $|G_i| = \lfloor \frac{|D_i|}{\beta} \rfloor$ and $G_i \subseteq D_i$.*

Lemma 5.2. *Step 2 can be done in $\lceil \log |D_i| \rceil + 1$ rounds.*

Lemma 5.3. *We can construct a transfer graph as described in Step 3 with in-degree at most 1 and out-degree at most $\beta - 1$.*

Lemma 5.4. *The in-degree of any disk in the transfer graph is at most β . The out-degree of any disk in the transfer graph is at most $2\beta - 2$. Moreover, the multiplicity of the graph is at most 4.*

Theorem 5.5. *The total number of rounds required for the multi-source multicast is $\max_i \lceil \log |D_i| \rceil + 3\beta + 3$.*

As the lower bound on the optimal number of rounds is $\max(\max_i \lceil \log |D_i| \rceil, \beta)$, we have a 4-approximation algorithm.

5.1 Allowing Bypass Nodes

The main idea is that without bypass nodes, only a small fraction of N disks is included in G_i for some i , if one disk requests many items while, on average, each disk requests few items. If we allow bypass nodes and hence G_i is not necessary a subset of D_i , we can make G_i very big so that each of almost all N disks belongs to some G_i . Bigger G_i reduces the out-degree of the transfer graphs and thus reduces the total number of rounds.

Algorithm Multi-source Multicast Allowing Bypass Nodes

1. We define $\bar{\beta}$ as $\frac{1}{N} \sum_{i=1 \dots N} |\{j | i \in D_j\}|$. In other words, $\bar{\beta}$ is the number of items a disk could receive, averaging over all disks. We arbitrarily choose a disjoint collection of subsets G_i , $i = 1 \dots \Delta$ with a constraint that $|G_i| = \lfloor \frac{|D_i|}{\bar{\beta}} \rfloor$. By allowing bypass nodes, G_i is not necessary a subset of D_i .
2. This is the same as Step 2 in the Multi-Source Multicast Algorithm, except that the source for item i (namely disk i) may belong to G_i for some i .
3. This step is similar to Step 3 in the Multi-Source Multicast Algorithm. We add $\lceil \bar{\beta} \rceil$ edges from each disk in G_i to satisfy $\lceil \bar{\beta} \rceil \cdot \lfloor \frac{|D_i|}{\bar{\beta}} \rfloor$ disks in D_i , and add at most another $\lceil \bar{\beta} \rceil - 1$ edges from disk i to satisfy the remaining disks in D_i .
4. This is the same as Step 4 in the Multi-Source Multicast Algorithm.

Theorem 5.6. *The total number of rounds required for the multi-source multicast algorithm, by allowing bypass nodes, is $\max_i \lceil \log |D_i| \rceil + \beta + \lceil 2\bar{\beta} \rceil + 6$.*

We now argue that $\lceil 2\bar{\beta} \rceil$ is a lower bound on the optimal number of rounds. Intuitively, on average, every disk has to spend $\bar{\beta}$ rounds to send data, and another $\bar{\beta}$ rounds to receive data. As a result, the total number of rounds cannot be smaller than $\lceil 2\bar{\beta} \rceil$. Allowing bypass node does not change the fact that $\max(\max_i \lceil \log |D_i| \rceil, \beta)$ is the other lower bound. Therefore, we have a 3-approximation algorithm.

References

1. E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan and J. Wilkes. An Experimental Study of Data Migration Algorithms. *Workshop on Algorithm Engineering*, 2001
2. B. Baker and R. Shostak. Gossips and Telephones. *Discrete Mathematics*, 2:191–193, 1972.
3. J. Bermond, L. Gargano and S. Perennes. Optimal Sequential Gossiping by Short Messages. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, Vol 86, 1998.
4. J. Bermond, L. Gargano, A. A. Rescigno and U. Vaccaro. Fast gossiping by short messages. *International Colloquium on Automata, Languages and Programming*, 1995.
5. S. Berson, S. Ghandeharizadeh, R. R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. *SIGMOD*, 1994.
6. J. A. Bondy and U. S. R. Murty. Graph Theory with applications. *American Elsevier*, New York, 1977.
7. R. T. Bumby. A Problem with Telephones. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):13–18, March 1981.
8. E. J. Cockayne, A. G. Thomason. Optimal Multi-message Broadcasting in Complete Graphs. *Utilitas Mathematica*, 18:181–199, 1980.
9. G. De Marco, L. Gargano and U. Vaccaro. Concurrent Multicast in Weighted Networks. *SWAT*, 193–204, 1998.

10. A. M. Farley. Broadcast Time in Communication Networks. *SIAM Journal on Applied Mathematics*, 39(2):385–390, 1980.
11. P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Mathematics*, 53:79–133, 1994.
12. L. Golubchik, S. Khanna, S. Khuller, R. Thurimella and A. Zhu. Approximation Algorithms for Data Placement on Parallel Disks. *Proc. of ACM-SIAM SODA*, 2000.
13. J. Hall, J. Hartline, A. Karlin, J. Saia and J. Wilkes. On Algorithms for Efficient Data Migration. *Proc. of ACM-SIAM SODA*, 620–629, 2001.
14. A. Hajnal, E. C. Milner and E. Szemerédi. A Cure for the Telephone Disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972.
15. S. M. Hedetniemi, S. T. Hedetniemi and A. Liestman. A Survey of Gossiping and Broadcasting in Communication Networks. *Networks*, 18:129–134, 1988.
16. I. Holyer. The NP-Completeness of Edge-Coloring. *SIAM J. on Computing*, 10(4):718–720, 1981.
17. J. Hromkovic and R. Klasing and B. Monien and R. Peine. Dissemination of Information in Interconnection Networks (Broadcasting and Gossiping). *Combinatorial Network Theory*, pp. 125–212, D.-Z. Du and D.F. Hsu (Eds.), Kluwer Academic Publishers, Netherlands, 1996.
18. C. A. J. Hurkens. Spreading Gossip Efficiently. *Nieuw Archief voor Wiskunde*, 5(1):208–210, 2000.
19. S. Kashyap and S. Khuller. Algorithms for Non-Uniform Size Data Placement on Parallel Disks. *Manuscript*, 2003.
20. S. Khuller, Y. A. Kim and Y. C. Wan. Algorithms for Data Migration with Cloning. *ACM Symp. on Principles of Database Systems* (2003).
21. W. Knodel. New gossips and telephones. *Discrete Mathematics*, 13:95, 1975.
22. H. M. Lee and G. J. Chang. Set to Set Broadcasting in Communication Networks. *Discrete Applied Mathematics*, 40:411–421, 1992.
23. D. Liben-Nowell. Gossip is Synteny: Incomplete Gossip and an Exact Algorithm for Syntenic Distance. *Proc. of ACM-SIAM SODA*, 177–185, 2001.
24. C. H. Papadimitriou. Computational complexity. *Addison-Wesley*, 1994.
25. D. Richards and A. L. Liestman. Generalizations of Broadcasting and Gossiping. *Networks*, 18:125–138, 1988.
26. H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29:442–467, 2001.
27. H. Shachnai and T. Tamir. Polynomial time approximation schemes for class-constrained packing problems. *Proc. of Workshop on Approximation Algorithms*, 2000.
28. C.E. Shannon. A theorem on colouring lines of a network. *J. Math. Phys.*, 28:148–151, 1949.
29. M. Stonebraker. A Case for Shared Nothing. *Database Engineering*, 9(1), 1986.
30. R. Tijdeman. On a Telephone Problem. *Nieuw Archief voor Wiskunde*, 19(3):188–192, 1971.
31. V. G. Vizing. On an estimate of the chromatic class of a p-graph (Russian). *Diskret. Analiz.* 3:25–30, 1964.

Approximating the Achromatic Number Problem on Bipartite Graphs

Guy Kortsarz and Sunil Shende*

Department of Computer Science, Rutgers University, Camden, NJ 08102
{guyk,shende}@camden.rutgers.edu

Abstract. The achromatic number of a graph is the largest number of colors needed to legally color the vertices of the graph so that adjacent vertices get different colors and for every pair of distinct colors c_1, c_2 there exists at least one edge whose endpoints are colored by c_1, c_2 . We give a greedy $O(n^{4/5})$ ratio approximation for the problem of finding the achromatic number of a bipartite graph with n vertices. The previous best known ratio was $n \cdot \log \log n / \log n$ [12]. We also establish the first non-constant hardness of approximation ratio for the achromatic number problem; in particular, this hardness result also gives the first such result for bipartite graphs. We show that unless NP has a randomized quasi-polynomial algorithm, it is not possible to approximate achromatic number on bipartite graph within a factor of $(\ln n)^{1/4-\epsilon}$. The methods used for proving the hardness result build upon the combination of one-round, two-provers techniques and zero-knowledge techniques inspired by Feige et.al. [6].

1 Introduction

A *proper coloring* of a graph $G(V, E)$ is an assignment of colors to V such that adjacent vertices are assigned different colors. It follows that each color class (i.e. the subset of vertices assigned the same color) is an independent set. A k -coloring is one that uses k colors. A coloring is said to be *complete* if for every pair of distinct colors, there exist two adjacent vertices which are assigned these two colors. The *achromatic number* $\psi^*(G)$ of a graph G is the *largest* number k such that G has a complete k -coloring.

A large body of work has been devoted to studying the achromatic number problem which has applications in clustering and network design (see the surveys by Edwards [4] and by Hughes and MacGillivray [11]). Yannakakis and Gavril [15] proved that the achromatic number problem is NP-hard. Farber et.al. [5] show that the problem is NP-hard on bipartite graphs. Bodlaender [1] established that the problem is NP-hard on graphs that are simultaneously co-graphs and interval graphs. Cairnie and Edwards [2] show that the problem is NP-hard on trees.

* Research supported in part under grant no. 9903240 awarded by the National Science Foundation.

Given the intractability of solving the problem optimally (assuming, of course, that $P \neq NP$), the natural approach is to seek a *guaranteed* approximation to the achromatic number. An *approximation algorithm* with ratio $\alpha \geq 1$ for the achromatic number problem is an algorithm that takes as input a graph G and produces a complete coloring of G with at least $\psi^*(G)/\alpha$ colors in time polynomial in the size of G . Let n denote the number of vertices in graph G . We will use the notation ψ^* for $\psi^*(G)$ when G is clear from the context.

Chaudhary and Vishwanathan [3] gave the first sublinear approximation algorithm for the problem, with an approximation ratio $O(n/\sqrt{\log n})$. Kortsarz and Krauthgamer [12] improve this ratio slightly to $O(n \cdot \log \log n / \log n)$. It has been conjectured [3] that the achromatic number problem *on general graphs* can be approximated within a ratio of $O(\sqrt{\psi^*})$. The conjecture is partially proved in [3] with an algorithm that gives a $O(\sqrt{\psi^*}) = O(n^{7/20})$ ratio approximation for graphs with *girth* (length of the shortest simple cycle) at least 7. Krysta and Lorys [13] give an algorithm with approximation ratio $O(\sqrt{\psi^*}) = O(n^{3/8})$ for graphs with girth at least 6. In [12], the conjecture is proved for graphs of girth 5 with an algorithm giving an $O(\sqrt{\psi^*})$ ratio approximation for such graphs. In terms of n , the best ratio known for graphs of girth 5 is $O(n^{1/3})$ (see [12]).

From the direction of lower bounds on approximability, the first (and only known) hardness of approximation result for general graphs was given in [12], specifically that the problem admits no $2 - \epsilon$ ratio approximation algorithm, unless $P=NP$. It could be that no $n^{1-\epsilon}$ ratio approximation algorithm (for any constant $\epsilon > 0$) is possible for general graphs (unless, say, $P=NP$). An $\Omega(n^{1-\epsilon})$ inapproximability result does exist for the maximum independent set problem [8] and the achromatic number problem and the maximum independent set problem are, after all, closely related.

On another negative note, consider the minimum maximal independent set problem. A possible “greedy” approach for finding an achromatic partition is to iteratively remove from the graph maximal independent sets of small size (maximality here is with respect to containment). However, the problem of finding a *minimum maximal independent set* cannot be approximated within ratio $n^{1-\epsilon}$ for any $\epsilon > 0$, unless $P=NP$ [7].

To summarize, large girth (i.e. girth greater than 4) is known to be a sufficient condition for a relatively low ratio approximation to exist. It is not known if the absence of triangles helps in finding a good ratio approximation algorithm for the problem. All the current results thus point naturally to the next frontier: the family of bipartite graphs.

1.1 Our Results

We give a combinatorial greedy approximation algorithm for the achromatic number problem on bipartite graphs achieving a ratio of $O(n^{4/5})$ and hence breaking the $\tilde{O}(n)$ barrier (the upper bound for general graphs [12]). We also give a hardness result that is both the first non-constant lower bound on approximation for the problem on general graphs, and the first lower bound on approximation for bipartite graphs. We prove that unless $NP \subseteq RTIME(n^{\text{poly} \log n})$,

the problem does not admit an $(\ln n)^{1/4-\epsilon}$ ratio approximation, for any constant $\epsilon > 0$.

This improves the hardness result of 2 on *general graphs* [12]. Note that the result in [12] constructs a graph with large cliques, which therefore is not bipartite. The best previous hardness result for bipartite graphs was only the NP-hardness result [5].

2 Preliminaries

We say that a vertex v is *adjacent* to a set of vertices U if v is adjacent to at least one vertex in U . Otherwise, v is *independent* of U . Subsets U and W are adjacent if for some $u \in U$ and $w \in W$, the graph has the edge (u, w) . U *covers* W if every vertex $w \in W$ is adjacent to U . We note that in a complete coloring of G , every pair of distinct color classes are adjacent to each other. For any subset U of vertices, let $G[U]$ be the subgraph of G induced by U . A *partial* complete coloring of G is a complete coloring of some induced subgraph $G[U]$, namely, a coloring of U such that all color classes are pairwise adjacent. Lemmas 1 and 2 below are well known [3,4,13,14]:

Lemma 1. *A partial complete coloring can be extended greedily to a complete coloring of the entire graph.*

Lemma 2. *Consider v , an arbitrary vertex in G , and let $G \setminus v$ denote the graph resulting from removing v and its incident edges from G . Then, $\psi^*(G) - 1 \leq \psi^*(G \setminus v) \leq \psi^*(G)$.*

A collection M of edges in a graph G is called a *matching* if no two edges in M have a common endpoint. The matching M is called *semi-independent* if the edges (and their endpoints) in M can be indexed as $M = \{(x_1, y_1), \dots, (x_k, y_k)\}$ such that both $X = \{x_1, \dots, x_k\}$ and $Y = \{y_1, \dots, y_k\}$ are independent sets, and for all $j > i$, it holds that x_i is *not* adjacent to y_j . As a special case, if x_i is not adjacent to y_j for all i, j , then the matching is said to be *independent*. A semi-independent matching can be used to obtain a partial complete coloring, as demonstrated in the next lemma; a weaker version, based on an independent matching, is used in [3].

Lemma 3. [14] *Given a semi-independent matching of size $\binom{t}{2}$ in a graph, a partial complete t -coloring of the graph (i.e. with t color classes) can be computed efficiently.*

Now, consider a presentation of a bipartite graph $G(U, V, E)$ with independent sets U and V forming the bipartition, and with edges in E connecting U and V . Assume that U has no isolated (degree 0) vertices. If $\Delta(V)$, the largest degree of a vertex in V , is suitably small, then by repeatedly removing a *star* formed by the current largest degree vertex in V and its neighbors in U , we can obtain a collection of at least $|U|/\Delta(V)$ stars. By picking a single edge out of every star, we get a semi-independent matching of size at least $|U|/\Delta(V)$. Applying Lemmas 1 and 3, we get the following result.

Lemma 4. *Let $G(U, V, E)$ be a bipartite graph with no isolated vertices in U . Then, the star removal algorithm produces an achromatic partition of size at least $\Omega(\sqrt{|U|/\Delta(V)})$.*

Hell and Miller [9,10] define the following equivalence relation (called the reducing congruence) on the vertex set of G (see also [4,11]). Two vertices in G are *equivalent* if and only if they have the same set of neighbors in the graph. We denote by $S(v, G)$, the subset of vertices that are equivalent to v under the reducing congruence for G ; we omit G when it is clear from the context. Assume that the vertices of G are indexed so that $S(v_1), \dots, S(v_q)$ denote the equivalence classes of vertices, where q is the number of distinct equivalence classes. Note that two equivalent vertices cannot be adjacent to each other in G , so $S(v_i)$ forms an independent set in G . The *equivalence graph* (also called the *reduced graph*) of G , denoted G^* , is a graph whose vertices are the equivalence classes $S(v_i)$ ($1 \leq i \leq q$) and whose edges connect $S(v_i), S(v_j)$ whenever the set $S(v_i)$ is adjacent to the set $S(v_j)$.

Lemma 5. [12] *A partial complete coloring of G^* can be extended to a complete coloring of G . Hence, $\psi^*(G) \geq \psi^*(G^*)$.*

Theorem 1. [12] *Let G be a bipartite graph with q equivalence classes of vertices. Then, there is an efficient algorithm to compute an achromatic partition of G of size at least $\min\{\psi^*/q, \sqrt{\psi^*}\}$. Thus, the achromatic number of a bipartite graph can be approximated within a ratio of $O(\max\{q, \sqrt{\psi^*}\})$.*

Let the *reduced degree* $d^*(v, G)$ be the degree of the vertex $S(v)$ in the reduced graph G^* ; equivalently, this is the maximum number of pairwise non-equivalent neighbors of v . Then, we have:

Lemma 6. *Let v, w be a pair of vertices of G such that $S(v) \neq S(w)$ and $d^*(w) \geq d^*(v)$. Then there is a vertex z adjacent to w but not to v .*

Proof. Assume that every neighbor of w is also a neighbor of v . Since $d^*(w) \geq d^*(v)$, it follows that v and w have exactly the same set of neighbors contradicting $S(w) \neq S(v)$. $\square$

3 The Approximation Algorithm

Let $\psi^*(G)$ denote the maximum number of parts in an achromatic partition of a graph G (we omit G in the notation when the graph is clear from the context). We may assume that ψ^* is known, e.g. by exhaustively searching over the n possible values or by using binary search. Throughout, an algorithm is considered efficient if it runs in polynomial time. Let $G(U, V, E)$ be a bipartite graph, and consider subsets $U' \subseteq U$ and $V' \subseteq V$. The (bipartite) subgraph induced by U' and V' is denoted by $G[U', V']$ where the (implicit) edge set is the restriction of E to edges between U' and V' .

Our approach is to iteratively construct an achromatic partition of an induced subgraph of $G[U, V]$. Towards this end, we greedily remove a *small, independent*

set of vertices A_i in each iteration while also deleting some other vertices and edges. The invariant maintained by the algorithm is that A_i always has a U -vertex and the subset of U vertices that survive the iteration is *covered* by A_i . This ensures that the collection $\mathcal{A} = \{A_i : i \geq 1\}$, forms a partial complete coloring of G .

To obtain such a collection $\mathcal{A}$ with large cardinality, we need to avoid deleting too many *non-isolated* vertices during the iterations since the decrease in achromatic number may be as large as the number of deleted, non-isolated vertices (by Lemma 2, while noting that the achromatic number remains unchanged under deletions of isolated vertices). For every $i \geq 1$, consider the sequence of induced subgraphs of G over the first $(i - 1)$ iterations, viz. the sequence $G_0 \supset G_1 \dots \supset G_{i-1}$ where G_k , $0 \leq k < i$, is the surviving subgraph at the beginning of the $(k + 1)^{th}$ iteration. The algorithm uses the following notion of *safety* for deletions in the i^{th} iteration:

Definition 1. *During iteration i , the deletion of some existing set of vertices S from G_{i-1} is said to be **safe** for G_{i-1} if the number of non-isolated vertices (including those in S) cumulatively removed from the initial subgraph G_0 is at most $\psi^*(G)/4$.*

3.1 Formal Description of the Algorithm

We first provide a few notational abbreviations that simplify the formal description and are used in the subsequent analysis of the algorithm. A set is called **heavy** if it contains at least $n^{1/5}$ vertices. Otherwise, it is called **light**. Given a subset of vertices U belonging to graph G , we denote by $d^*(v, U, G)$ the maximum number of pairwise non-equivalent neighbors that v has in U . v is called **U -heavy** if $d^*(v, U, G) \geq n^{1/5}$.

The approximation algorithm ABIP described below produces an achromatic coloring of G . It invokes the procedure PARTITION (whose description follows that of ABIP) twice, each time on a different induced subgraph of G . The procedure returns a partial complete achromatic partition of its input subgraph.

Algorithm ABIP.

Input: $G(U, V)$, a bipartite graph.

1. Let $\mathcal{A}_1 = \text{PARTITION}(G[U, V])$, and let $G^{[1]} = G[U^{[1]}, V^{[1]}]$ be the induced subgraph that remains when the procedure halts.
2. Let $\mathcal{A}_2 = \text{PARTITION}(G[V^{[1]}, U^{[1]}])$; note that the roles of the bipartitions are *interchanged*. Let $G^{[2]} = G[U^{[2]}, V^{[2]}]$ be the induced subgraph that remains when this second application halts.
3. If either of the achromatic partitions $\mathcal{A}_1$ or $\mathcal{A}_2$ is of size at least $\psi^*/(16 \cdot n^{1/5})$, then the corresponding partial complete coloring is extended to a complete achromatic coloring of G which is returned as final output.
4. Otherwise, apply the algorithm of Theorem 1 on the subgraph $G^{[2]}$. A partial complete coloring is returned which can then be extended to a complete achromatic coloring of G returned as final output.

Procedure PARTITION**Input:** $G_0(U_0, V_0)$, an induced subgraph of a bipartite graph $G(U, V)$.

1. **if** $\psi^* < 8 \cdot n^{4/5}$, **return** an arbitrary achromatic partition.
2. $\mathcal{A} \leftarrow \{\}$ /* $\mathcal{A}$ contains the collection of A_i 's computed so far */
3. **for** $i = 1, 2, \dots$ /* Iteration i */
 - a) **if** there are no light U_{i-1} -equivalence classes in G_{i-1} , then **break**
 - b) Choose a vertex $u \in U_{i-1}$ with smallest equivalence class size and smallest reduced degree in G_{i-1} (break ties arbitrarily).
 - c) Remove $S(u)$ from U_{i-1} , the neighbors of u from V_{i-1} and let $G' = G[U', V']$ be the resulting induced subgraph.
 $C_i \leftarrow \emptyset$
 - d) **while** $U' \neq \emptyset$ and there exists a U' -heavy vertex in V' **do**
 - i. Choose v with largest reduced degree $d^*(v, U', G')$ in the current graph G' .
 - ii. Add v to C_i
 - iii. Remove v from V' and its neighbors from U'
 - e) Let q' be the number of U' -equivalence classes in G'^* .
 - f) **if** $q' > n^{3/5}$, let $\mathcal{A}$ be the partition obtained by applying the star removal algorithm to G' (see Lemma 4).
break
 - g) Let D_i be the vertices in U' with light equivalence classes in G'
 - h) **for** every heavy U' -equivalence class $S(w)$ **do**
 add an arbitrary neighbor of $S(w)$ to C_i .
 - i) $A_i \leftarrow S(u, G_{i-1}) \cup C_i$;
 Let $L_i \subseteq V_{i-1}$ be the set of isolated vertices in the graph $G[U_{i-1} \setminus A_i, V_{i-1} \setminus A_i]$
 - j) **if** it is **not safe** to delete $(A_i \cup D_i)$ from G_{i-1} then **break**
 - k) add A_i to $\mathcal{A}$;
 Remove $S(u, G_{i-1})$ and D_i from U_{i-1} leaving U_i
 Remove C_i and L_i from V_{i-1} leaving V_i
 $G_i \leftarrow G[U_i, V_i]$
4. **return** $\mathcal{A}$

3.2 The Approximation Ratio

We now analyze the approximation ratio of ABIP; detailed proofs have been omitted due to space considerations. Our goal is to show that the approximation ratio is bounded by $O(n^{4/5})$. The analysis is conducted under the assumption that $\psi^*(G) \geq 8 \cdot n^{4/5}$. Otherwise, returning an arbitrary achromatic partition (say, the original bipartition of size 2), as done in line 1 of PARTITION, trivially gives an $O(n^{4/5})$ ratio.

We start by observing that the loop on line 3 in PARTITION could exit in one of *three mutually exclusive ways* during some iteration $(k+1) \geq 1$. We say that PARTITION takes

- **exit 1** if the star removal algorithm can be applied (at line 3f) during the iteration,
- **exit 2** if just prior to the end of the iteration, it is found that the current deletion of $(A_{k+1} \cup D_{k+1})$ is not safe for G_k (at line 3j), or
- **exit 3** if at the beginning of the iteration, there are no light U_k -equivalence classes in G_k (at line 3a).

Note that the induced subgraphs G_i ($i \geq 1$) form a monotone decreasing chain so if the star removal algorithm (exit 1) is not applicable at any intermediate stage, then PARTITION will eventually take one of the latter two exits, i.e. the procedure *always* terminates. We say that iteration $i \geq 1$ in an execution of PARTITION is *successful* if none of the exits are triggered during the iteration, i.e. the procedure continues with the the next iteration. Let $(k+1) \geq 1$ be the first unsuccessful iteration.

Lemma 7. *If PARTITION takes exit 1 during iteration $(k+1)$, then the achromatic partition returned has size at least $n^{1/5}$. As $\psi^* \leq n$, an $O(n^{4/5})$ -ratio is derived.*

Proof. Consider U' and V' when exit 1 is triggered. It is easy to show that every vertex $w \in U'$ is adjacent to V' . Furthermore, the inner loop condition (line 3d) guarantees that every vertex in V' is adjacent to at most $n^{1/5}$ U' -equivalence classes. When the star removal algorithm is applied, q' (the number of U' -equivalence classes) is at least $n^{3/5}$. From the discussion preceding Lemma 4, it is easy to see that the star removal algorithm will produce a collection of at least $\sqrt{n^{3/5}/n^{1/5}} = n^{1/5}$ stars. Thus, an achromatic partition of size at least $n^{1/5}$ is returned as claimed. $\square$

Next, we show that if the procedure takes exit 2 during iteration $(k+1)$ because an unsafe deletion was imminent, then it must be the case that k is large and hence, that we have already computed a large achromatic partition $\mathcal{A} = \{A_1, A_2, \dots, A_k\}$. To this end, we establish a sequence of claims.

Claim 1 *For all i such that $1 \leq i \leq k$, the set A_i is an independent set and is adjacent to A_j for every $j \in [i+1, k]$. Equivalently, $\mathcal{A}$ is an achromatic partition of the subgraph $G[\cup_{1 \leq i \leq k} A_i]$.*

Proof. We first verify that at the end of a successful iteration i , the set of vertices A_i is an independent set. By construction, the vertices retained in U_i at the end of the iteration are all covered by C_i . The set A_j , for $i < j \leq k$, contains at least one vertex in $U_{j-1} \subset U_i$. Hence there is always an edge between A_i and A_j . $\square$

Claim 2 *For all i such that $1 \leq i \leq k$, the size of the set $(A_i \cup D_i)$, just prior to executing the safety check on line 3j, is bounded by $4n^{4/5}$.*

Proof. By construction, $A_i = S(u) \cup C_i$ prior to executing line 3j. We know that $S(u)$ is a light equivalence class and hence, $|S(u)| < n^{1/5}$. A vertex $v \in V_{i-1}$ is added to C_i either during the inner loop (line 3d) or later, if it happens to

be adjacent to a heavy U' -equivalence class (line 3h). In both cases, we can show that no more than $n^{4/5}$ vertices could have been added to C_i . Together, we have at most $3 \cdot n^{4/5}$ being added to A_i . Now, U' has at most $n^{3/5}$ light equivalence classes when control reaches line 3g. Since the vertices in D_i just prior to executing the safety check are simply those belonging to such light U' -equivalence classes, there are at most $n^{1/5} \cdot n^{3/5} = n^{4/5}$ vertices in D_i . $\square$

Claim 3 *If the first k iterations are successful, then the difference, $\psi^*(G_0) - \psi^*(G_k)$, is at most $4k \cdot n^{4/5}$.*

Proof. Follows from Lemma 2 and Claim 2. $\square$

Lemma 8. *If PARTITION takes exit 2 during iteration $(k + 1)$, then the achromatic partition returned has size at least $\lfloor \psi^*(G)/16n^{4/5} \rfloor$ thus giving an $O(n^{4/5})$ ratio approximation.*

Proof. Since the first k iterations were successful, it follows that for each $i \in [1, k]$, it is safe to delete $(A_i \cup D_i)$. However, it is unsafe to delete $(A_{k+1} \cup D_{k+1})$ and by Definition 1 and Claim 3, this can only happen if

$$4(k + 1)n^{4/5} > \psi^*(G)/4.$$

Hence $\mathcal{A} = \{A_1, A_2, \dots, A_k\}$, which is an achromatic partition of the subgraph $G[\cup_{1 \leq i \leq k} A_i]$ by Claim 1, has size $k \geq \lfloor \psi^*(G)/(16n^{4/5}) \rfloor$. Applying Lemma 1, we conclude that a complete achromatic coloring of G with at least $\lfloor \psi^*(G)/(16n^{4/5}) \rfloor$ colors can be computed thus giving an $O(n^{4/5})$ ratio approximation. $\square$

Finally, if PARTITION takes exit 3 in iteration $(k + 1)$, then we have two possibilities. If $k \geq \lfloor \psi^*(G)/(16n^{4/5}) \rfloor$, a sufficiently large partition has been found and we are done. Otherwise, $k < \lfloor \psi^*(G)/(16n^{4/5}) \rfloor$ and we may not necessarily have a good approximation ratio. However, note that G_k , the graph at the beginning of iteration $(k + 1)$, has no light U_k -equivalence classes (this triggers the exit condition). Hence, U_k has no more than $n^{4/5}$ equivalence classes that are all heavy, since each heavy class has at least $n^{1/5}$ vertices and $|U_k| \leq n$.

Claim 4 *Assume that both applications of PARTITION on lines 1 and 2 of algorithm ABIP respectively take exit 3. Let q_1 (respectively, q_2) be the number of light $U^{[1]}$ -equivalence classes in $G^{[1]}$ (respectively, the number of light $U^{[2]}$ -equivalence classes in $G^{[2]}$). Then, the graph $G^{[2]}$ has achromatic number at least $\psi^*(G)/2$ and at most a total of $(q_1 + q_2) \leq 2n^{4/5}$ equivalence classes.*

Proof. Observe that the removal of vertices (along with all their incident edges) from a graph cannot increase the number of equivalence classes: two vertices that were equivalent before the removal, remain equivalent after. Hence, the number of $V^{[2]}$ equivalence classes is at most q_1 (note that the partitions are interchanged before the second application of PARTITION on line 2). Thus $G^{[2]}$

has at most a total of $(q_1 + q_2)$ equivalence classes. The discussion preceding the claim shows that $(q_1 + q_2)$ is bounded above by $2n^{4/5}$. Since neither application of PARTITION took exit 2, the vertices deleted during both applications were safe for deletion. Hence, the net decrease in the achromatic number is at most $2\psi^*(G)/4 = \psi^*(G)/2$. $\square$

Theorem 2. *The algorithm ABIP has an approximation ratio of $O(n^{4/5})$.*

Proof. By Lemmas 7 and 8, if either of the two applications of PARTITION take exits 1 or 2, then we are guaranteed a ratio of $O(n^{4/5})$. If both applications of PARTITION on lines 1 and 2 of ABIP halt on exit condition 3, then an application of the algorithm of Theorem 1 on graph $G^{[2]}$ (see line 4 of ABIP) provides an $O(q)$ approximation ratio for $G^{[2]}$ where q is the number of equivalence classes of $G^{[2]}$. By claim 4, this achromatic coloring is a partial complete coloring of G with ratio $O(n^{4/5})$. $\square$

4 A Lower Bound for Bipartite Graphs

Let $G(U, V, E)$ be a bipartite graph. A *set-cover* (of V) in G is a subset $S \subseteq U$ such that S covers V , i.e. every vertex in V has a neighbor in S . Throughout, we assume that the intended bipartition $[U, V]$ is given explicitly as part of the input, and that every vertex in V can be covered. A *set-cover packing* in the bipartite graph $G(U, V, E)$ is a collection of pairwise-disjoint set-covers of V . The *set-cover packing problem* is to find in an input bipartite graph (as above), a maximum number of pairwise-disjoint set-covers of V . Our lower bound argument uses a modification of a construction by Feige et.al. [6] that creates a set-cover packing instance from an instance of an NP-complete problem. Details of the construction are omitted due to space limitations and will appear in the full version of the paper. The main result obtained is the following:

Theorem 3. *For every $\epsilon > 0$, if NP does not have a (randomized) quasi-polynomial algorithm then the achromatic number problem on bipartite graphs admits no approximation ratio better than $(\ln n)^{1/4-\epsilon}/16$.*

4.1 The Set-Cover Packing Instance [6]

Our lower bound construction uses some parts of the construction in [6]. That paper gives a reduction from an arbitrary NP-complete problem instance I to a set-cover packing instance¹ $G(U, V, E)$, with $|U| + |V| = n$.

The idea is to use a collection of disjoint sets of vertices $\{A_i : 1 \leq i \leq q\}$ and $\{B_i : 1 \leq i \leq q\}$; all these sets have the same size N where N is a parameter. In the construction, $U = (\bigcup_{i=1}^q A_i) \cup (\bigcup_{i=1}^q B_i)$. Also, the set $V = \bigcup M(A_i, B_j)$ with the union taken over certain pre-defined pairs (A_i, B_j) that arise from the NP-complete instance. The set $M(A_i, B_j)$ is called a *ground-set*. The reduction uses randomization to specify the set of edges E in the bipartite graph with the following properties:

¹ The construction described here corresponds to the construction in [6] for the special case of two provers.

1. If I is a yes-instance of the NP-complete problem, then U can be decomposed into N vertex-disjoint set-covers $S_1, \dots, S_N$ of V . Each set-cover contains a unique A vertex and a unique B vertex for every A, B . Each S_i is an *exact cover* of V in the sense that every V -vertex has a unique neighbor in S_i .
2. In the case of a no-instance, the following properties hold:
 - a) **The A, B property:** Only the $A_i \cup B_j$ vertices are connected in G to $M(A_i, B_j)$. **Comment:** The next properties concern the induced sub-graph $G[(A_i \cup B_j), M(A_i, B_j)]$.
 - b) **The random half property:** Each $a \in A_i$ and $b \in B_j$ is connected in $M(A_i, B_j)$ to a random half of $M(A_i, B_j)$.
 - c) **The independence property:** For every $a \in A_i$ and $b \in B_j$, the collection of neighbors of a in $M(A_i, B_j)$ and the collection of neighbors of b in $M(A_i, B_j)$ are mutually independent random variables.
 - d) **The equality or independence property:** The neighbors of two vertices $a, a' \in A_i$ in $M(A_i, B_j)$ are either the same, or else their neighbors in $M(A_i, B_j)$ are mutually independent random variables. A similar statement holds for a pair of vertices $b, b' \in B_j$.

Thus, vertices $a \in A_i$ and $b \in B_j$ have, on average, $|M(A_i, B_j)|/4$ common neighbors in $M(A_i, B_j)$ because a and b are joined to two independent random halves in $M(A_i, B_j)$.

4.2 Our Construction

The basic idea is similar to the above construction, namely that we wish to convert a **yes** instance of the NP-complete problem to a bipartite graph with a “large” achromatic partition and a **no** instance to a bipartite graph with a “small” achromatic partition. Towards this end, we extend the construction in [6] as follows.

Construction of a yes instance: A *duplication* of a vertex $u \in U$ involves adding to U a new copy of u connected to the neighbors of u in V . By appropriately duplicating the original vertices in U , we can make the number of vertex-disjoint set-covers larger. Specifically, we can duplicate vertices in U to ensure that every A and B set has $|V|$ elements and hence, the number of set-covers in the packing for a **yes** instance becomes $|V|$ as well (recall, from the previous discussion, that for a **yes** instance, each set-cover contains exactly one A vertex and exactly one B vertex, and so $|A| = |B| = |V|$ is the number of set-covers in the packing).

Using some technical modifications, we can also make G regular. Hence, G admits a perfect matching where each $v \in U$ is matched to some corresponding vertex $m(v) \in V$. Observe that for the case of a yes instance, the number of $m(v)$ vertices, namely, $|V|$, is equal to the number of set-covers in the set-cover packing.

The idea now is to form a collection of $|V|$ sets, one for each $v \in U$, by adding the matched vertex $m(v)$ to an (exact) set-cover S_i . However, the resulting sets

are not independent sets because each S_i is an exact set-cover of V and hence contains a neighbor of $m(v)$. But this problem can be fixed by ensuring that during the duplication process, a *special* copy of v is inserted; specifically, the special copy gets all the neighbors of v *except* $m(v)$ as its neighbors. With this modification, the collection consists of independent sets which form an achromatic partition because each of the S_i 's are exact. This implies that in the case of a yes instance, the corresponding bipartite graph admits a size $|V|$ complete coloring.

Construction of a no instance: The main technical difficulty is showing that in a case of a no-instance, the maximum size achromatic partition is "small". Let $X_1, X_2, X_3, \dots$ be the color classes in a maximum coloring in the case of a no-instance. Consider the contribution of A, B to the solution, i.e. how many vertices from the A and B sets belong to any X_i . Observe that in the case of a yes instance, each color contains one vertex from every A and every B . If we could color the graph corresponding to a no instance with "many" colors, this would mean that each X_i has to contain only "few" A and "few" B vertices. Similarly, for a yes instance, each color contains exactly one V vertex. Therefore, each X_i must contain only "few" $M(A, B)$ vertices.

Say, for example, that for every i , X_i satisfies the conditions: $|X_i \cap M(A, B)| = 1$ and $|X_i \cap (A \cup B)| = 1$ as in a yes instance. Let $v_2 \in (X_i \cap M(A, B))$ and $v_1 \in (X_i \cap (A \cup B))$. Observe that the random half property implies that the edge (v_1, v_2) exists only with probability $1/2$.

On the other hand, we note that if a coloring has close to $|V|$ colors, events as the one above should hold true for $\Omega(|V|^2)$ pairs. since every X_i and X_j must by definition share at least one edge. The equality or independence property ensures that "many" (but not all) of events such as the ones above are independent. Therefore, it is unlikely that polynomially many such independent events can occur simultaneously.

The above claim has its limits. If we take subsets of size, say, $\sqrt{2 \log n}$ from $A \cup B$ and $M(A, B)$ into every X_i, X_j , then the number of "edge-candidates" between X_i and X_j is now $(\sqrt{2 \log n})^2 = 2 \log n$. Namely, each one of the $\log n$ pairs is a possible candidate edge, so that if it is chosen by the randomized choice, it guarantees at least one edge between X_i and X_j as required by a legal achromatic partition. Every candidate edge exists with probability $1/2$. Thus the probability for at least one edge between X_i and X_j could be as high as $1 - 1/n^2$, and it is not unlikely that "many" (like $|V|^2 < n^2$) of these events happen simultaneously.

Note this if each X_i contains roughly $\sqrt{\log n}$ vertices from every A, B and from every $M(A, B)$, the number of colors in the solution could be as high as $|V|/\sqrt{\log n}$. This gives a limitation for this method (namely, we can not expect a hardness result beyond $\sqrt{\log n}$). In fact, the hardness result we are able to prove is only $(\log n)^{1/4-\epsilon}$ due to the fact that the events described above are not really totally independent.

Acknowledgments. The first author would like to thank Robert Krauthgamer and Magnus M. Halldórsson for useful discussions and comments.

References

1. H. L. Bodlaender. Achromatic number is NP-complete for cographs and interval graphs. *Inform. Process. Lett.*, 31(3):135–138, 1989.
2. N. Cairnie and K. Edwards. Some results on the achromatic number. *J. Graph Theory*, 26(3):129–136, 1997.
3. A. Chaudhary and S. Vishwanathan. Approximation algorithms for the achromatic number. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 558–563, 1997.
4. K. Edwards. The harmonious chromatic number and the achromatic number. In *Surveys in combinatorics, 1997 (London)*, pages 13–47 .
5. M. Farber, G. Hahn, P. Hell, and D. Miller. Concerning the achromatic number of graphs. *J. Combin. Theory Ser. B*, 40(1):21–39, 1986.
6. U. Feige, M. Halldórsson, G. Kortsarz and A. Srinivasan. Approximating the domatic number. Accepted to Siam J. on Computing conditioned on a revision.
7. M. M. Halldórsson. Approximating the minimum maximal independence number. *Inform. Process. Lett.*, 46(4):169–172, 1993.
8. J. Hastad, Clique is Hard to Approximate within n to the power $1-\epsilon$, *Acta Mathematica*, Vol 182, 1999, pp 105–142.
9. P. Hell and D. J. Miller. On forbidden quotients and the achromatic number. In *Proceedings of the 5th British Combinatorial Conference (1975)*, pages 283–292. Congressus Numerantium, No. XV. Utilitas Math., 1976.
10. P. Hell and D. J. Miller. Achromatic numbers and graph operations. *Discrete Math.*, 108(1-3):297–305, 1992.
11. F. Hughes and G. MacGillivray. The achromatic number of graphs: a survey and some new results. *Bull. Inst. Combin. Appl.*, 19:27–56, 1997.
12. G. Kortsarz and R. Krauthgamer. On approximating the achromatic number. *Siam Journal on Discrete Mathematics*, vol 14, No. 3, pages: 408–422
13. P. Krysta and K. Loryś. Efficient approximation algorithms for the achromatic number. In *ESA '99 (Prague)*, pages 402–413. Springer, 1999.
14. A. Máté. A lower estimate for the achromatic number of irreducible graphs. *Discrete Math.*, 33(2):171–183, 1981.
15. M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM J. Appl. Math.*, 38(3):364–372, 1980.

Adversary Immune Leader Election in ad hoc Radio Networks^{*}

Mirosław Kutylowski^{1,2} and Wojciech Rutkowski¹

¹ Institute of Mathematics, Wrocław University of Technology,
`{mirekk,rutkow}@im.pwr.wroc.pl`
² CC Signet

Abstract. Recently, efficient leader election algorithms for ad hoc radio networks with low time complexity and energy cost have been designed even for the no-collision detection, (no-CD), model. However, these algorithms are fragile in the sense that an adversary may disturb communication at low energy cost so that more than one station may regard itself as a leader. This is a severe fault since the leader election is intended to be a fundamental subroutine used to avoid collisions and to make it reliable.

It is impossible to make the leader election algorithm totally resistant – if an adversary is causing collisions all the time no message comes through. So we consider the case that an adversary has limited energy resources, as any other participant.

We present an approach that yields a randomized leader election algorithm for a single-hop no-CD radio network. The algorithm has time complexity $O(\log^3 N)$ and energy cost $O(\sqrt{\log N})$. This is worse than the best algorithms constructed so far ($O(\log N)$ time and $O(\log^* N)$ energy cost), but succeeds in presence of an adversary with energy cost $\Theta(\log N)$ with probability $1 - 2^{-\Omega(\sqrt{\log N})}$. (The $O(\log^* N)$ energy cost algorithm can be attacked by an adversary with energy cost $O(1)$.)

1 Introduction

Radio Network Model. The networks considered in this paper consist of processing units, called *stations*, which communicate through a shared communication channel. Since neither the number of stations nor their ID's are known, they are described as *ad hoc networks*. Since a shared communication channel may be implemented by a radio channel, they are called *radio networks*, or *RN* for short.

The networks of this kind are investigated quite intensively due to several potential applications, such as sensor networks [5]. If deploying a wired network is costly (e.g. a sensor network for collecting and analyzing certain environment data) or impossible (e.g. an ad hoc network of stations deployed in cars for optimization of road traffic), such networks might be the only plausible solution.

In this paper we consider *single-hop* RN's: if a station is sending a message any other station may receive it (so we concern networks working on a small area). If two

^{*} partially supported by KBN grant 7 T11C 032 20

stations are sending simultaneously then a *collision* occurs. We make here a (standard) pessimistic assumption that in the case of a collision the messages are scrambled beyond recognition. Moreover, we assume that the result is indistinguishable from a random noise – the model is called *no-CD RN*. In accordance with industrial standards, we also assume that a station cannot simultaneously transmit and listen.

Unlike in other distributed systems, RN's work synchronously and computation is divided into *time slots*. (It has been pointed that it is a realistic assumption - at least if GPS signals can be received by the stations.) In a time slot a station can be either transmitting or listening or may perform internal work only. In the first two cases we say that a station is *awake*.

Usually, the stations of a RN are battery operated. So it is crucial to design energy efficient algorithms - it helps to eliminate failures due to battery exhaustion. Since energy consumption of a processor is lower by an order of magnitude than the energy consumption for transmitting and listening, we consider only energy consumption due to communication. Unexpectedly, in the case of small networks the energy for transmitting and listening are comparable [5], so we can disregard these differences until the final fine tuning of the algorithms. Quality of an RN algorithm is determined usually by two complexity measures: time – the number of time slots used; and energy cost – the maximum of the number of steps over all stations during which a station is awake.

In the literature different scenarios are considered regarding the stations and their knowledge: either the number of stations is known, or the number of stations is known up to some multiplicative constant, or the stations have no knowledge about the number of other active stations. A similar situation is for the station ID's: either the stations have no ID's at all (and they have to be assigned during so called initialization), or in the range $1..n$, where the number of active stations is $\Omega(n)$.

Further information concerning ad hoc networks can be found in a handbook [10].

Leader Election Problem. Most algorithms running on RN's in a multiprocessor environment assign some special roles to certain processing stations. The simplest kind of such an initialization is the following *Leader Election Problem*:

A RN network is given, such that each active station has no knowledge which stations are active (except itself). Initialize the network so that *exactly* one station gets status *leader* and the rest of the active stations receive the status *non-leader*.

Additional assumptions may be considered: the stations may know approximately the number of active stations, the stations may have unique ID's in a given range $1..n$. (Note that if the stations know that all ID's in this range are used, the problem is trivial: the station with ID 1 is the leader. The point is that a station is not aware of the other stations: in particular it must consider the case in which all stations with low ID are non-active or do not exist.)

Security issues. Practical applications of RN's must take into account various dangers resulting from the communication model. Although in a wired network an adversary attacking the system might inject packets with manipulated data concerning their origin, it is possible to trace their origin to some extent. In wireless networks it is much harder.

As long as no special physical equipment is deployed, the mobile intruder is safe. The algorithms built so far for RN's disregard this issue. This is a severe drawback, since for many applications ad hoc networks must offer a certain security level to be useful.

Security problems of previous solutions. The simplest leader election algorithm is used by Ethernet [9]: with a certain probability each candidate sends a message (we call it a *trial*). If a station sends a message and there is no collision, then it becomes a leader. The step is repeated until the leader is elected. However, this solution demands that every participant must listen to or send a message all the time, so energy cost equals execution time. Another disadvantage is that even if the expected runtime is $O(1)$, getting the leader with high probability is not that easy [11]. The main problem from our point of view is that the adversary may cause collisions all the time – in this way its energy cost will remain the same as the cost of stations trying to elect the leader.

Another algorithm of this kind is presented in [8] – it is assumed that the number of active stations is unknown, the stations may detect collisions, and that the algorithm must be uniform, i.e. all stations perform *trials* using the same probabilities depending on computation history (but not on processor ID). It is easy to disguise this algorithm with extra collisions. Namely, the algorithm uses collisions for adjusting probabilities. Fake collisions may cause overestimating the number of stations electing the leader and consequently executing the *trial* with a wrong probability such that nobody sends.

If the stations have unique ID's in the range $1..n$ (with some ID numbers unused), then leader election may be performed deterministically by the following simple *Tree Algorithm* [7]: Consider a binary tree with n leaves labeled 1 through n . In this tree a leaf j represents the station with ID j . The internal nodes of the tree have unique labels $n+1, n+2, \dots$ so that the label of a parent is bigger than the labels of its child nodes. The leader of the subtree rooted at node $n+i$ is found at step i : The leader of the subtree rooted in the left child node of node $n+i$ sends a message and the leader of the subtree rooted in the right child node of node $n+i$ is listening. (If there is no active station in a subtree, then there is no leader of the subtree and nobody sends or listens, respectively.) If such a message is sent, then the leader of the right subtree becomes a non-leader. Otherwise it considers itself as a leader of the subtree with the root at node i . The leader of the left subtree becomes the leader of the subtree rooted at node i whenever it exists.

Tree Algorithm is not immune: For instance, it suffices to make a collision at some point where there are leaders of the right and left subtrees of a given node. Then the leader of the right subtree would think that the left subtree is empty and will regard itself as the leader of both subtrees. The same is assumed by the leader of the left subtree. Starting from this moment both stations will behave exactly the same and make a collision whenever they send. This leads even to an avalanche so that more and more leaders emerge. Paper [7] presents an algorithm with Tree Algorithm executed as a final stage. In particular, this part can be attacked.

In paper [6] another strategy is applied for the model where collisions are detectable. In the first phase *trials* are performed with probabilities $2^{-2^{i^2}}$ for $i = 1, 2, \dots$ until there is no collision, say for $i = T$. The stations that have not transmitted are no longer considered candidates for the leader. During the second phase *trials* are executed with probabilities 2^{-2^i} for $i = T, T-1, \dots$. Again, in the case of a collision all stations that have

not transmitted are eliminated. This tricky algorithm can be efficiently attacked by an adversary. An adversary who knows the approximate number of stations electing the leader simulates collisions so that $2^{2^{T^2}} > n^2$ - only $O(1)$ additional steps are required. Then, all candidates perform the *trial* with probability $2^{-2^{T^2}} < n^{-2}$. So with a high probability no candidate for the leader sends. However, the adversary may send a collision signal. In this case all candidates stop trying to become the leader (in the original algorithm it makes sense, they know that the stations that caused the collision remain.) The attack requires $O(\sqrt{\log \log n})$ energy cost, while the original algorithm has energy cost $O(\log \log n)$. The adversary can achieve that no leader is elected.

In [4] three energy efficient leader election algorithms are presented. The first of them - a log star deterministic algorithm - working for special inputs executes Tree Algorithm with an efficient reassignment of tasks to stations. Anyway, it is at least as vulnerable as Tree Algorithm. The third algorithm from [4] uses the first algorithm as a final stage. It is also insecure.

The general deterministic leader election algorithm from [4] achieves energy cost $(\log n)^{o(1)}$. It is based on Tree Algorithm, but can be attacked in one more way. The algorithm splits the candidates into groups and elects the leader of each group in two phases: in phase 1 each active participant of the group transmits - if there is no collision then the only active participant knows that it is a leader. Otherwise, we execute recursively the algorithm within the group: an advantage is that the leader elected for this group gets at least one slave - this slave is used afterwards to reduce the energy cost of its master. An adversary causes a collision during phase 1. So the costly phase 2 is executed, but the leader gets no slave.

New Result. Our main goal is to design a leader election algorithm that is energy efficient and tolerates an adversary having limited energy resources. The secondary goal is to preserve polylogarithmic execution time. We get the following result:

Theorem 1. *Consider a single-hop no-CD radio network consisting of $O(n)$ stations sharing a secret key. Assume that the stations are not initialized with ID's. Then it is possible to elect a leader of the network with energy cost $O(\sqrt{\log N})$ within time $O(\log^3 N)$, so that the outcome is faulty with probability $O(2^{-\sqrt{\log N}})$ in the presence of the adversary station which has energy cost $O(\log N)$.*

2 Basic Techniques

Cryptographic Assumptions. We assume that the stations that are electing the leader have a common secret s entirely hidden from the adversary. How to deploy such a secret is a problem considered in the literature as *key predistribution problem*. In the simplest case the stations are equipped with these secrets at the time of personalization by inserting secret data stored on smart cards (on the other hand, it is unpredictable which of them will actually participate in the protocol).

Secret s can be used for protecting the protocol: messages sent at time step t can be XOR-ed with a string $s_t = f(s, t)$, where f is a secure pseudorandom function. This

makes the messages transmitted indistinguishable from a random noise. For this reason the adversary cannot detect when and which messages have been transmitted by the stations electing a leader.

Using such stream encipherment is realistic, since the processor may prepare the pseudorandom strings in advance. This fits also quite well to the fact that the processor of the station is active much longer than the station is sending/receiving. The function f need not be very strong in a cryptographic sense – it is only required that it cannot be broken until a leader is elected or leak s . In order to protect a common secret s we may replace it by a value $s_0 = g(s, t_0)$, where g is a secure hash function and t_0 is, for instance, the starting moment of the election procedure.

Thanks to the use of cryptographic techniques we may assume that the knowledge of the adversary is confined to the knowledge of the algorithm executed, its starting point and the approximate number of stations electing the leader. It cannot derive any information from the communication. So the adversary may only hope to cause collisions at right moments and in this way to break down the protocol.

Initial Selection. It is quite obvious that *trials* are well suited to select a small group of candidates for the leader in a way resistant to adversary attacks. Let us recall them in detail: By a *participant* we shall mean any station which performs the leader election protocol. Let us assume that there are $n = \Theta(N)$ participants and N is known to all stations (including the adversary). First a participant tosses a coin and with probability 0.5 it becomes a *sender*, otherwise it becomes a *receiver*. Then it participates in d rounds (d is a parameter): during a round a participant decides to be active with probability N^{-1} . The round consists of 3 communication steps: during step 1 an active sender broadcasts a random message, simultaneously each active receiver listens, at step 2 the receiver repeats the message received while the sender listens, if it gets back its own message, then it repeats it, while the receiver is listening. If the sender gets its own message, it knows that there was no collision and exactly one participant has responded. At step 3 the receiver may check whether there was no collision. In this case we say that a pair of participants, the sender and receiver, succeeds, and we assign to them the round number as a temporary ID.

In order to keep energy cost $O(1)$ we also assume that a participant may decide to be active for at most one round. Despite that the process is not a sequence of Bernoulli trials, one may prove that the number of successes does not change substantially [1].

Note that initial selection is resistant against an adversary: since we assume that its energy capacity is $O(\log N)$, for a sufficiently large $r = \Theta(\log N)$, the adversary can only reduce probability of success in a trial if r trials are performed.

Time Windows. Suppose that the stations with a common secret s have to exchange one message and a time window consisting of r consecutive steps is available for this purpose. Then, basing on the secret s they may determine the moment within the window in which they actually communicate (by transmitting ciphertexts). Then an adversary can neither detect a transmission nor guess the moment of transmission. So he can send messages blindly hoping to get a collision. So, if the adversary sends m messages within the window, the probability of a collision at transmission moment equals $\frac{m}{r}$.

There is a trade off between security protection and time complexity of the protocol: using windows of size r changes time complexity by a factor of r .

Random Reassignment of ID's. After an initial selection there are candidates that will elect a leader among themselves. An adversary can disturb choosing candidates for only a small fraction of ID numbers. The problem is that it can focus on some ID number (for instance consecutive ones), if it could bring some advantages to him. There is a simple countermeasure: each temporary ID u is replaced by $\pi_{s,t_0}(u)$, where π is a pseudorandom permutation $\pi_s = h(s, t_0)$, where h is an appropriate cryptographic generator of pseudorandom permutations based on seed s and t_0 .

3 Adversary Resistant Leader Election

The algorithm consists of a preprocessing and $v = \Theta(\sqrt{\log N})$ *group election* phases. Preprocessing chooses a relatively small group of candidates for the leader. Each group election phase has the goal to elect a leader from a different group of candidates chosen by preprocessing assigned to this phase. There is no election of the leader from the group leaders – the first group that succeeds in choosing a group leader “attacks” all subsequent group election phases so that it prevents electing a leader from any subsequent group.

Preprocessing. Initial selection from Section 2 is executed for $d = v \cdot k$, where $k = O(\log N)$. If a round j of initial selection succeeds, then the stations chosen are denoted by P_j and P'_j , and called, *stations with ID j* . The station P_j is a *candidate* for the leader in our algorithm, P'_j is required for auxiliary purposes.

In this way we establish $\Omega(d)$ pairs of stations with ID's in the range $1..d$. The pairs with ID's $(i-1) \cdot k, \dots, i \cdot k$, are assigned for group election phase i . Before assigning the ID's permutation technique (Section 2) is executed. So an adversary attacking during the initial selection cannot determine the ID's that are eliminated by the collisions.

A Group Election Phase. Let us consider the ID's assigned to this phase. Since every round of the initial selection may fail, we talk about *used* and *unused* ID's - in the later case there is no candidate for the group leader with this ID.

Let us arrange all ID's assigned to this phase in a line. Some of these ID's are used, however they may be separated by blocks of unused ID's. At the beginning of a phase each candidate of the group knows that its own ID is used but has no idea about the status of the other ID's. The goal of the phase is to chain all used ID's of this group. First each used ID has to find the next used ID according to the natural ordering. Then each candidate will get knowledge about the whole chain. Then it will be straightforward to determine a group leader by applying a certain deterministic function to the chain - this can be done locally by each candidate.

For the sake of simplicity of exposition we assume that the ID's assigned to the phase considered are 1 through k . Building a chain without energy cost limitations is easy: at step j a candidate P_a such that $a < j$ and there is no candidate P_l with ID $l \in (a, j)$, sends a message and P_j listens. If P_j responds at the next step, then P_a knows its successor

in the chain and stops its activity of building the chain. If there is no response, then it assumes that ID j is unused and proceeds to step $j + 1$ in which it tries to contact P_{j+1} . However, this approach has certain drawbacks. First, it may happen that there is a long block of unused ID's. In this case the last candidate before the block would use a lot of energy to find a subsequent used ID. The second problem is that an adversary may cause collisions - so it might happen that the knowledge of different candidates becomes inconsistent. It is easy to see that in this case the result of the algorithm would be faulty.

Building the chain consists of two sub-procedures. During the first one we build disconnected chains. The second sub-procedure has the goal to merge them. The next two sections present the details.

In the last subsection we describe how to modify these sub-procedures so that a group that succeeds in choosing a group leader can prevent electing a leader in all subsequent groups (note that the changes must be done carefully, these capabilities cannot be granted for an adversary!).

Building Chains. The procedure of building chains uses k communication slots, each one consisting of four windows (Section 2) of size $\Theta(\log^{3/2} N)$. Each chain corresponds to an interval of ID's inspected by the stations related to the chain. At each moment a chain has an *agent*, which is the station P_a , where a is the last used ID in the interval mentioned. The last ID in the interval (it need not be a used ID) is the *end of the chain*. During the execution of the protocol the chains grow, and at communication slot j , we try to attach ID j into a chain ending at position $j - 1$. There are two potential parties active in this trial: stations P_j, P'_j and stations P_a, P'_a , where P_a is the agent of a chain ending at position $j - 1$. The last chain is called *the current chain*. (For $j = 1$ there is no chain below so there are no agents, but the stations may execute the same code.) For the purpose of communication, the information about the current chain can be encoded by a string of length k : it contains a 1 at position j , if j is a used ID, and a 0, if j is an unused ID, or symbol $-$, if position j does not belong to the chain yet. During communication slot j four steps are executed, during which P_j, P'_j, P_a, P'_a are listening whenever not transmitting.

Step 1: The agent P_a of the current chain transmits the string encoding status of the current chain.

Step 2: P'_a repeats the message from the previous step.

Step 3: P_j acknowledges that it exists.

Step 4: P'_j repeats the message of P_j received in the previous step.

After exchanging these messages the stations have to make decisions concerning their status. If there is no adversary, the following situations may happen. If there are agents of the current chain, ID j is *used* and the communication has not been scrambled, then position j joins the chain and P_j, P'_j take over the role of agents of the current chain. If no proper message is received at the first and the second step, then P_j, P'_j start a new chain. If no message is received at the last two steps, then agents of the current chain assume that the ID j is unused and retain the status of the agents, position j joins the chain as an unused ID. However, there is one exception: if energy usage of P_a, P'_a approaches the bound $b = O(\sqrt{\log N})$, they loose the status of the agents. Since no other

stations take over, the current chain terminates in this way. Additionally, P_a , P'_a reach status *last-in-a-chain*.

Now we consider in detail what happens if an adversary scrambles communication (this is a crucial point since no activity of the adversary should yield an inconsistent view of the stations).

Case I: the 3rd message is scrambled. Since P_j does not receive its own message back, P_j and P'_j are aware of the attack. They must take into account that the agents of the current chain may hear only a noise and conclude that the ID j is unused. In order to stay consistent with them, P_j and P'_j “commit a suicide” and from this moment do not participate in the protocol. So the situation becomes the same as in the case they have not existed at all.

Case II: the 4th message is scrambled, and the 3rd one is not. Then P_j behaves as in Case I. Only P'_j is unaware of the situation. However, it will never hear its partner P_j , so will never respond with a message (only sending such a message could do some harm.)

In Cases I and II the agents of the current chain either hear only a noise or at most one unscrambled message. (The first case occurs also when the ID j is unused.) They know that ID j is unused or the sender with ID j commits a suicide. Therefore the agents retain their status of agents of the current chain.

Case III: the last two messages are not scrambled and at least one message of the first two is scrambled. In this case the agent P_a does not receive its message back. Also the stations P_j , P'_j are aware that either there was no agent or at least one message was scrambled. In this case P_j decides to start a new chain. At the same time P_a decides to terminate the current chain and reaches the status *last-of-the-chain*. Additionally, P_j can also acknowledge to P_i that its transmission was clean of an adversary if the attacker decides to attack the second message.

Note that a chain may terminate for two reasons. First, there can be a large block of unused ID's so that the agents exhaust their energy resources. However, we shall see that this case happens with a small probability. The second case is that an adversary causes starting of a new chain (Case III) or enforces suicides (Cases I and II). In the later case the agents of the current chain do not change, which may consequently lead to energy exhaustion. However, we shall see that this may happen with a quite small probability. In the only remaining case a new chain starts exactly one position after the last position of the previous chain. This makes merging the chains easy.

Merging Chains. Before we start this part of execution each ID number j is replaced by $((j - 1 + f(s, i)) \bmod k) + 1$, where f is a cryptographic one-way function and s is the common secret of the stations electing the leader. A slight disadvantage is that in this way we split one chain, namely the chain containing information on ID's $k - f(s, i)$ and $k - f(s, i) + 1$. However, the advantage is that the adversary cannot attack easily the same places as during the building chains phase.

The procedure of merging chains takes $O(\log N)$ communication slots, each consisting of two windows of size $\Theta(\log^{3/2} N)$. For each chain, all its members know the first used ID of the chain. If it is t , then we can also label the whole chain by t .

Consider a chain t . Our goal is to merge it with other chains during communication slots $t - 1$ and t . With a high probability the phase of building chains yields a chain ending at position $t - 1$. During the merging procedure this chain may be merged with other chains. So assume that immediately before communication slot $t - 1$ a chain l ends at position $t - 1$. At this moment the following invariant holds: *each candidate with ID in chain l knows status of all ID's in this chain, each candidate P_j of chain t knows the status of all ID's except the ID's larger than the first used ID j' larger than j* . Then:

slot $t - 1$: the member of chain l with status *last-in-a-chain* transmits a message encoding the status of all ID's in this chain, all candidates from chain t listen,

slot t : all candidates from chains l and t listen, except the station in chain t with status *last-in-a-chain*, which sends a message encoding status of all ID's in chain t .

In order to improve resistance against an adversary these two steps can be repeated a couple of times (this makes the time and energy cost grow). It is easy to see that if the messages come through, then all stations from chains l and t are aware of ID's in both chains. So, in fact, these chains are merged and the invariant holds.

If an adversary makes a collision at this moment, then the chains l and t do not merge and in subsequent communication slots chain t grows while chain l remains unchanged.

After merging the chains we hopefully have a single chain. Even if not, it suffices that there is a chain having information about at least $k/2$ ID's and corresponding to $c \cdot k$ candidates (where c is a constant) - of course there is at most one such a chain in a group. Finally, the members of this chain elect a group leader in a deterministic way based on secret s and information about used ID's in the chain.

3.1 Disabling the Later Groups

The idea of the “internal attack” against electing a group leader is to prevent emerging too large chains in the sub-procedure of building chains and preventing merging these chains during the second sub-procedure so that no resulting chain has length at least $k/2$. The attack is performed by a group of $w = \Theta(\sqrt{\log N})$ stations that succeeded in electing a group leader assigned to attack this particular group. Each of them will be involved in communication only a constant number of times. (So $\Omega(\log N)$ candidates from the group with the group leader are enough and their energy cost grows only by an additive constant.)

The first modification is that each group contains w special positions that are reserved for *alien candidates*, not belonging to the group. The positions of aliens are located in w intervals of length k/w , one alien per interval. The precise location of an alien position within the interval depends on the secret s in a pseudo-random way.

While electing a group leader the alien positions are treated as any other positions in the group. However they may be “served” only by the stations from a group that has already elected a group leader. So if no group leader has been elected so far, then “an alien ID” work as an unused ID and the election works as before. During an attack the “alien stations” P_j and P'_j corresponding to alien position j behave as follows: when position j is considered they scramble step 1 (or 2) and perform correctly steps 3 and 4. So the agent of the current chain decides to stop the chain, making place for the chain

starting at position j . When position $j + 1, j + 2, \dots$ are considered the alien stations do not transmit (as it would be the case if they adhered to the protocol). So no chain starting at position j is built, in the best case the next chain starts at position $j + 1$.

When the chains of the group are merged, then there is no chance to merge the chains separated by at least one position - no further intervention of the aliens is necessary. So the aliens chop the chains into pieces of small size. It is easy to see that at least $w/2 - 1$ consecutive aliens must fail so that a chain of length greater than $k/2$ is built.

Let us discuss what can be done by an adversary to disable the internal attack (this would lead to the election of multiple group leaders). The only chance of the adversary is to make collisions at steps 3 and 4, when alien stations P_j and P'_j send messages according to the protocol (consequently, Case I or II and not Case III will apply and the current chain will not terminate at this point). The chances of the adversary are not big since it must guess correctly the alien positions and guess proper positions in the windows when these messages are transmitted. Finally, disabling separation of the chain at one place is not enough - the adversary must succeed in so many places that a chain of length at least $k/2$ emerges.

4 Proof of Correctness

Algorithm Behavior with no Adversary. First we check that the algorithm succeeds with a fair probability, if there is no adversary. Except for the “internal attack” the only reason for which a group may fail to elect its leader is that there is a block of unused ID’s of size at least $b = \Omega(\sqrt{\log N})$. (In fact, this is a necessary condition, but not a sufficient one, a long block of unused ID’s close to the first or to the last position would not harm.) Indeed, otherwise each agent P_a encounters a used ID j for $j < a + b$, so before energy limit b of P_a is reached. Then P_j becomes the agent of the same chain and, in particular, the chain does not terminate. We see that if there are no large blocks of unused ID’s a single chain is constructed. It contains all ID’s except the unused ID’s in front of the first used ID (there are less than b of them according to our assumption). Merging chains in this case does not change anything. Finally, there is a chain of size at least $k - b > k/2$ so it elects the group leader.

Let us estimate the probability of creating a large block of unused ID’s in a group. As noticed in Section 2, the probability of success in a single Ethernet trial is at least μ , where $\mu \approx \frac{1}{e^2}$. There is a technical problem since the success probabilities in different trials are not independent, but according to [3, Lemma 2] we can bound the probability of getting a block of unused ID’s of length m by probability of m successes in a row for independent Bernoulli trials with success probability $1 - \mu$.

There are $k - b$ positions where a block of unused ID’s of length at least b may start. The probability of emerging a block at such a point is at most $(1 - \mu)^b$. So we can upper bound the probability that there is at least one block of at least b unused ID’s by $(1 - \mu)^b \cdot (k - b)$. Since $b = \Theta(\sqrt{\log N})$, $k = \Theta(\log N)$ this probability is $2^{-\Omega(\sqrt{\log N})}$.

Algorithm Behavior in Presence of an Adversary. In order to facilitate the estimation of an error probability we assume that the adversary may have energy cost $z = O(\log N)$ during each computation part analyzed below.

Initial Selection. The adversary can only lower the probability of success in Ethernet trials. Due to the random reassignment of ID's (as described in Section 2) the adversary does not know which ID it is attacking by making collisions during Ethernet trials. The reassignment permutation is pseudorandom, hence we may assume that a given ID is made unused by the adversary with probability at most $z/k < 1$. So the probability that a given ID becomes unused (whatever the reason is) is upper bounded by $(1 - \mu) + z/k$. If we adjust constants in a right way, then the last expression is a constant less than 1 and we may proceed as in the previous subsection to show that with probability at most $2^{-\Omega(\sqrt{\log N})}$ a block of at least b unused ID's can emerge.

Building chains. Now we can assume that there is no block of unused ID's of length at least b . So the adversary must prevent construction of a large chain. For this purpose, he must break a chain during the first part (Section 3) and prevent merging the chains at this position later. The next lemma estimates the chances of the adversary.

Lemma 1. *The probability of breaking the chains during the building chain procedure and preventing them from merging in at least one point during a single group election phase is bounded by a constant less than 1.*

Proof. From the point of view of an adversary breaking the chains is a game: first he chooses (at most) z out of k positions (these are positions attacked during the building chain procedure). The next move in the game is performed by the network: there is a secret pseudorandom shift of positions. The distance of the shift remains unknown for the adversary. In the next move the adversary chooses again (at most) z positions. The adversary loses, if no position previously chosen is hit again (since there is no place where the chains are not merged again). Even if the adversary succeeds in attacking a place where the network tries to merge the chains, it must make collisions in proper places in two time windows: even if the whole energy z is invested in a single window, then the success probability (for the adversary) in this window is $O(1/\sqrt{\log N})$. So the probability of succeeding in two independent windows is $O(1/\log N)$.

Let p_i be the probability of hitting by the adversary i positions during the third move of the game. In order to win, the adversary has to prevent merging the chains in at least one of these positions. Probability of such an event is $O(i \cdot \frac{1}{\log N})$. Finally, the total probability that the adversary wins the game is

$$O\left(\sum_{i=1}^z p_i \cdot \left(i \cdot \frac{1}{\log N}\right)\right). \quad (1)$$

Now consider all k shifts for fixed sets of z positions chosen by the adversary during the first and during the third move. Let us count the number of hits. Each position chosen during the third move hits each position chosen during the first move for exactly one shift. So the total number of hits equals z^2 . On the other hand, the number of hits can be counted by the expression $\sum_{i=1}^z (p_i \cdot k) \cdot i$. It follows that $\sum_{i=1}^z p_i \cdot i = \frac{z^2}{k} = O(\log N)$. Hence the probability given by expression (1) is bounded by a constant α less than 1. $\square$

Note that in order to prevent electing a leader the adversary has to succeed in every group. It happens with probability $O(\alpha^v)$ which is $O(2^{-\sqrt{\log N}})$, since computations in different groups may be regarded as independent.

Disabling “Internal Attacks”. There is still another chance for an adversary: He can assume that in at least one round the leader was chosen, and attack one of the following group election phases. His goal is to disable the internal attack of the group that already has a leader. In this way multiple leader emerge as a result of an algorithm execution. For this purpose the adversary has to guess at least $\frac{w}{2} - 1$ “alien positions”. For each of these positions, he has to guess the right time slots inside two windows and collide twice with the “aliens”. The probability of such an event is less than:

$$\left(\frac{w}{2} - 1\right) \cdot \left(\left(\frac{1}{\sqrt{\log N}}\right)^2\right)^{w/2-1} = 2^{-\Omega(\sqrt{\log N})}.$$

5 Final Remarks

Our algorithm offers some additional features – it yields a group of $\Omega(\log N)$ active stations which know each other. This can turn out to be useful for replacing a leader that gets faulty or exhausts its energy resources.

References

1. Jurdziński, T., Kutylowski, M., Zatópiański, J.: Energy-Efficient Size Approximation for Radio Networks with no Collision Detection. COCOON’2002, LNCS 2387, Springer-Verlag, 279–289
2. Jurdziński, T., Kutylowski, M., Zatópiański, J.: Weak Communication in Radio Networks. Euro-Par’2002, LNCS 2400, Springer-Verlag, 965–972
3. Jurdziński, T., Kutylowski, M., Zatópiański, J.: Weak Communication in Single-Hop Radio Networks – Adjusting Algorithms to Industrial Standards. full version of [2], to appear in Concurrency and Computation: Practice & Experience
4. Jurdziński, T., Kutylowski, M., Zatópiański, J.: Efficient Algorithms for Leader Election in Radio Networks. ACM PODC’2002, 51–57
5. Estrin, D.: Sensor Networks Research: in Search of Principles. invited talk at PODC’2002, www.podc.org/podc2002/estrin.ppt
6. Nakano, K., Olariu, S.: Randomized Leader Election Protocols for Ad-Hoc Networks. SIROCCO’2000, Carleton Scientific, 253–267
7. Nakano, K., Olariu, S.: Randomized Leader Election Protocols in Radio Networks with no Collision Detection. ISAAC’2000, LNCS 1969, Springer-Verlag, 362–373
8. Nakano, K., Olariu, S.: Uniform Leader Election Protocols for Radio Networks. ICPP’2001, IEEE
9. Metcalfe, R. M., Boggs, D. R.: Ethernet: Distributed Packet Switching for Local Computer Networks. Communication of the ACM 19 (1976), 395–404
10. Stojmenović, I. (Ed.): Handbook of Wireless Networks and Mobile Computing. Wiley 2002
11. Willard, D.E.: Log-logarithmic Selection Resolution Protocols in Multiple Access Channel. SIAM Journal on Computing 15 (1986) , 468–477

Universal Facility Location

Mohammad Mahdian¹ and Martin Pál^{2*}

¹ Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA.

`mahdian@theory.lcs.mit.edu`

² Computer Science Department, Cornell University, Ithaca, NY 14853.

`mpal@cs.cornell.edu`

Abstract. In the Universal Facility Location problem we are given a set of demand points and a set of facilities. The goal is to assign the demands to facilities in such a way that the sum of service and facility costs is minimized. The service cost is proportional to the distance each unit of demand has to travel to its assigned facility, whereas the facility cost of each facility i depends on the amount of demand assigned to that facility and is given by a cost function $f_i(\cdot)$. We present a $(7.88 + \varepsilon)$ -approximation algorithm for the Universal Facility Location problem based on local search, under the assumption that the cost functions f_i are nondecreasing. The algorithm chooses local improvement steps by solving a knapsack-like subproblem using dynamic programming. This is the first constant-factor approximation algorithm for this problem. Our algorithm also slightly improves the best known approximation ratio for the capacitated facility location problem with non-uniform hard capacities.

1 Introduction

In the facility location problem, we are given a set of demands and a set of possible locations for facilities. The objective is to open facilities at some of these locations and connect each demand point to an open facility in such a way that the total cost of opening facilities and connecting demand points to open facilities is minimized.

Many variants of the problem have been studied, such as uncapacitated facility location, in which we pay a fixed cost for opening each facility and an open facility can serve any number of clients, hard-capacitated facility location, in which each facility has an upper bound on the amount of demand it can serve, or soft-capacitated facility location, in which each facility has a capacity but we are allowed to open multiple copies of each facility. Facility location problems have occupied a central place in operations research since the early 60's [2,12,14,20,17,5,7]. Many of these problems have been studied extensively from the perspective of approximation algorithms.

Linear Programming (LP) based techniques have been applied successfully to uncapacitated and soft-capacitated variants of the facility location problem to obtain constant factor approximation algorithms for these problems (See [19]

* Research supported in part by ONR grant N00014-98-1-0589.

for a survey and [15,16] for the latest results). However, LP based techniques have not been successful when dealing with hard capacities, and all known LP formulations have an unbounded integrality gap.

Using local search, Korupolu, Plaxton and Rajamaram [13] and Chudak and Williamson [6] gave a constant approximation algorithm for the hard capacitated problem, under the assumption that all capacities are the same. Pál, Tardos and Wexler [18] gave a local search algorithm for facility location with arbitrary hard capacities that achieves approximation ratio $9 + \varepsilon$. Local search heuristics have been successfully used for other variants of facility location [14,13,6,4,1].

In this paper we consider a generalized version of the facility location problem in which the cost of opening each facility is an arbitrary given function of the amount of demand served by it. This problem, which we call the *universal facility location problem*, was defined and studied for the case of concave facility cost functions in [10]. Great number of well-studied problems such as uncapacitated facility location, capacitated facility location with soft and hard capacities, facility location with incremental facility costs (a.k.a. the linear-cost facility location problem), and concave-cost facility location are special cases of the universal facility location problem. In this paper we present the first constant factor approximation algorithm for the universal facility location problem with non-decreasing facility costs. Our algorithm is a natural generalization of the local search algorithm of Pál et al. [18], and achieves an approximation factor of $8 + \varepsilon$. This slightly improves the best approximation factor known for the hard-capacitated facility location problem. Furthermore, since our algorithm employs local transformations that are more powerful than the operation of Charikar and Guha [4] for uncapacitated facility location, as well as the transformations of Arya et al. [1] for soft-capacitated facility location, our algorithm inherits approximation guarantees of both algorithms in these special cases. In other words, our algorithm provides a common generalization of several local search algorithms that have been proposed for various facility location problems.

The rest of this paper is organized as follows. In Section 2 we define the universal facility location problem, introduce some notation and mention some important special cases of universal facility location. In Section 3, we describe the overall structure of the algorithm, define the local operations and show how to find them efficiently. In Section 4, the analysis, we prove the approximation guarantee of the algorithm. Section 5 contains some concluding remarks.

2 Definitions

In this section, we first introduce the universal facility location problem, and then define several well-studied facility location problems as special cases of the universal facility location problem.

2.1 The Universal Facility Location Problem

The *Universal Facility Location* (UniFL) Problem is defined as follows. We are given a set D of *demand points* (a.k.a. *clients*), a set F of *facilities*, and a

weighted graph G on the vertex set $F \cup D$ with connection costs c on the edges. Each client $j \in D$ has d_j units of demand to be served by the facilities. The goal is to allocate certain capacity u_i at every facility $i \in F$ and assign all demands to the facilities subject to the constraint that each facility i can serve no more demand than the capacity u_i that has been allocated at it. We allow for splittable demands, that is, the demand of a client can be split among multiple facilities.

The cost $c(S)$ of a solution S to a UniFL instance is the sum of facility and service costs, denoted by $c_f(S)$ and $c_s(S)$, respectively. There is a cost associated with allocating capacity at each facility, depending on the amount allocated. Let us denote the cost of allocating u_i units of capacity at facility i by $f_i(u_i)$. We refer to the sum of costs of allocating capacity at all facilities as the *facility cost*. There is also a cost c_{ij} for shipping each unit of demand from client j to facility i . We assume that these connection costs form a metric, i.e., they are symmetric and obey the triangle inequality. The total shipping cost is referred to as the *service cost*.

A solution S to a UniFL instance can be specified by a pair (u, x) , where u is the *allocation vector* (i.e., for $i \in F$, u_i is the capacity allocated at facility i), and x is the *assignment vector* (i.e., x_{ij} denotes the amount of demand of client j served by facility i). Using this notation, we can write the universal facility location problem as the following nonlinear optimization problem.

$$\begin{aligned}
 & \text{minimize} && \sum_{i \in F} f_i(u_i) + \sum_{i \in F, j \in D} c_{ij} x_{ij} \\
 & \text{subject to} && \sum_{i \in F} x_{ij} = d_j && \forall j \in D \\
 & && \sum_{j \in D} x_{ij} \leq u_i && \forall i \in F \\
 & && u_i, x_{ij} \geq 0 && \forall i \in F, j \in D.
 \end{aligned}$$

Note that it makes sense to assume that the functions $f_i(\cdot)$ are non-decreasing (otherwise we can always allocate more capacity if it costs us less).¹ To model hard upper bounds on capacities, we allow f_i to take the value $+\infty$. We make the following assumptions about the functions f_i .

Assumption 1 *Each f_i is a non-decreasing and left-continuous mapping from non-negative reals to non-negative reals with infinity. That is, $f_i : \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0} \cup \{+\infty\}$, $f_i(x) \leq f_i(y)$ for every $x \leq y$, and $\lim_{x \rightarrow u^-} f_i(x) = f_i(u)$ for every u .*

The above assumption is not only helpful in designing an approximation algorithm, it also guarantees the existence of a globally optimal solution.

Theorem 2. *Any instance of UniFL with the cost functions satisfying Assumption 1 has an optimal solution.*

¹ In order to define the problem for facility cost functions that are not non-decreasing, we need to change the second constraint of the optimization program to equality. There are certain cases where having a decreasing cost function is useful. See Section 5 for an example.

In addition to the Assumption 1, it usually makes sense to assume that for every i , $f_i(0) = 0$, that is, we do not pay for facilities we do not use. However, we do not need this assumption in our analysis.

2.2 Special Cases

The Universal facility location generalizes several important variants of the facility location problem that have been studied in the literature.

The *uncapacitated facility location* problem is a special case of UniFL problem where cost functions are of the form $f_i(u) = \bar{f}_i \cdot [u > 0]$. That is, every facility serving positive demand must pay the opening cost $\bar{f}_i$, and any open facility can serve an unlimited amount of demand. The best approximation algorithm known for this problem is LP-based and achieves an approximation ratio of 1.52 [15]. The best local search algorithm for this problem is due to Charikar and Guha [4], and achieves a factor of $3 + \varepsilon$ ($1 + \sqrt{2} + \varepsilon$ with scaling). It is easy to observe that the local steps we use in our algorithm generalize the local steps of Charikar and Guha [4], and hence their analysis also shows that our algorithm achieves the same approximation guarantees for the uncapacitated facility location problem.

The *capacitated facility location problem with soft capacities* has cost functions of the form $f_i(u) = \bar{f}_i \cdot \lceil u/\bar{u}_i \rceil$. In words, at each site i we can open an arbitrary number of “copies” of a facility at cost $\bar{f}_i$ each. Each copy can serve up to $\bar{u}_i$ units of demand. This problem is also known as the capacitated facility location problem with integer decision variables in the operations research literature [3]. The best approximation algorithm known for this problem is LP-based and achieves a factor of 2 [16]. Arya et al. [1] give a local search algorithm with an approximation guarantee of $4 + \varepsilon$ for this problem. Since our local steps are at least as powerful as theirs, their approximation guarantee carries over to our algorithm in the soft-capacitated case.

The *facility location problem with incremental costs* (a.k.a. the *linear-cost facility location problem*) is a special case of the UniFL problem in which the facility costs are of the form $f_i(u) = \bar{f}_i \cdot [u > 0] + \sigma_i \cdot u$, i.e., in addition to opening costs $\bar{f}_i$, there is also cost σ_i per unit of demand served. This can be easily reduced to the uncapacitated problem by increasing all c_{ij} distances by σ_i .

In the *concave-cost facility location problem*, cost functions $f_i(\cdot)$ are arbitrary concave functions. This problem has been studied in the operations research literature [7]. A concave function can be well approximated by a collection of linear functions from its upper linear envelope. This suggests that UniFL with arbitrary concave cost functions can be reduced to facility location with incremental costs, and further to uncapacitated facility location (See [10] for details). Without much effort we can show that our algorithm performs these reductions implicitly, hence the $3 + \varepsilon$ guarantee of [4] carries over to instances with concave facility cost as well.

In the *capacitated facility location problem with hard capacities* each facility has an opening cost $\bar{f}_i$ and a hard capacity $\bar{u}_i$. An open facility can serve up to $\bar{u}_i$ demand, and this cannot be exceeded at any cost. Hence the cost function is $f_i(u) = \bar{f}_i \cdot [u > 0] + \infty \cdot [u > \bar{u}_i]$. Hard capacitated facility location

is perhaps the most difficult special case of UniFL, in that it captures much of the hardness of UniFL with respect to approximation. The only known approximation algorithm for this problem is due to Pál, Tardos, and Wexler [18], and achieves an approximation factor of $8.53 + \varepsilon$. Our algorithm is an extension of their algorithm.

3 The Algorithm

The basic scheme of the algorithm is very simple. Start with an arbitrary feasible solution and repeatedly look for local transformations that decrease the cost. If no such operation can be found, output the current solution and stop. Otherwise, pick the operation that decreases the cost by the greatest amount, apply it to the current solution and continue with the modified solution.

This simple scheme guarantees that (if it stops), the algorithm arrives at a *locally optimal* solution, i.e., a solution that cannot be improved by any local transformation. In Section 4 we will show that any locally optimal solution with respect to our operations is a good approximation for the (globally) optimal solution. Our algorithm employs the following two types of local transformations.

- **add**(s, δ). Increase the allocated capacity u_s of facility s by δ , and find the minimum cost assignment of demands to facilities, given their allocated capacities. A variant of this operation has been considered by many previous papers [1,13,6,18]. This operation allows us to bound the service cost (Lemma 1).

The cost of this operation is $f_s(u_s + \delta_s) - f_s(u_s) + c_s(S') - c_s(S)$, where $c_s(S)$ and $c_s(S')$ indicate the service cost of the solution before and after the operation, respectively. These costs can be computed by solving a minimum cost network flow problem.

- **pivot**(s, Δ). This operation is a combination of the open and close operations of [18]. It is strictly more powerful, however, as it allows us to prove a somewhat stronger approximation guarantee of $8 + \varepsilon$.

In the operation **pivot**(s, Δ), we adjust the amount of demand served by each facility i by Δ_i . For a facility i with $\Delta_i < 0$ this means that we must ship $|\Delta_i|$ units of excess demand out of i . We gather all this excess demand at a location s , which is the *pivot* for the operation. Subsequently we distribute the demand gathered at s to facilities with $\Delta_i > 0$. Finally, we adjust the allocated capacity of each facility to be equal to the actual amount of demand served by the facility. Note that since the amount of demand in the system must be conserved, the operation is feasible only if $\sum_i \Delta_i = 0$.

Assuming that before the operation the allocated capacity u_i of each facility i was equal to the actual demand served by the facility, the cost of the **pivot**(s, Δ) operation can be estimated as $\sum_{i \in F} f_i(u_i + \Delta_i) - f_i(u_i) + c_{si} \cdot |\Delta_i|$.²

To obtain polynomial running time, we need to address two issues.

² We can do a little better when computing the cost of reassigning demand from a facility i to the pivot s . Since each unit of demand at i originated from some client

Significant improvements. We shall require that every local transformation we apply improves the cost $c(S)$ of the current solution S significantly, that is, by at least $\frac{\epsilon}{5n}c(S)$ for some $\epsilon > 0$. We call such an operation *admissible*. If no admissible operation can be found, the algorithm stops. Note that since the optimum cost $c(S^*)$ is a lower bound on the cost of any solution, the algorithm stops after at most $\frac{5n}{\epsilon} \ln \frac{c(S)}{c(S^*)}$ iterations. This means that the solution we output may be only approximately locally optimal, not a true local optimum. The bound on the cost of solutions that are only approximately optimal is only by an ϵ factor worse than the bound for true local optima.

Efficient operation finding. We need to show that in each iteration we can find an admissible operation in polynomial time, if one exists. We do not know how to find the best local operations efficiently, because even for very simple functions f_i (such as the functions with two steps arising from capacitated facility location), finding an optimal local step is NP-hard. However, we are able to find a local operation with cost within a small additive factor, say $\frac{\epsilon}{10n}c(S)$ of the best operation in polynomial time by discretizing the costs and solving a knapsack-like subproblem using dynamic programming (see the full version³ for details). This guarantees that if an admissible operation exists, our algorithm finds an operation that improves the cost by at least $\frac{\epsilon}{10n}c(S)$. This is still enough to drive the cost down sufficiently fast.

4 The Analysis

The goal of this section is to show that if for some solution $S = (u, x)$ our algorithm cannot find an operation that would significantly improve its cost, then the cost of S is not too large in comparison to the optimal solution S^* . Our analysis roughly follows the analysis in [18].

For the most of this section we shall assume that S in fact is a locally optimal solution, i.e. no operation can improve its cost. We show that the cost of any locally optimal solution S is within a factor of 8 of the cost of the optimal solution S^* . In the end of the section we extend this argument to yield a bound of $8 + \epsilon$ for solutions for which no significant improvement can be found.

4.1 Bounding the Service Cost

Any local search algorithm that has the **add** operation in its repertoire can guarantee low service cost. A variant of the following lemma has been first proved in [13] and [8], and in various versions it has become folklore of the field since. See the full paper for a proof.

j , instead of shipping the demand from i to s at cost c_{is} , we can reroute it directly from j to s at cost $c_{js} - c_{ji}$. In our analysis, it is enough to use the original estimate. However, to be able to claim that our algorithm generalizes the algorithms of [4,1], we need to use the refined cost estimate.

³ available from <http://www.cs.cornell.edu/people/mpal/papers>

Lemma 1. *The service cost $c_s(S)$ of a locally optimal solution $S = (u, x)$ is at most the cost of the optimal solution $c(S^*)$.*

4.2 Bounding the Facility Cost

The bulk of work we have to do in the analysis goes to proving a bound on the facility cost. To do this, we start with a solution S that is locally optimal with respect to the **add** operation, and hence has small service cost. We argue that if the facility cost of S is large, then there must be a **pivot** operation that improves the cost of S . In order to illustrate the technique, we start by imagining that instead of the **pivot** operation, we have a stronger global operation **swap**, as defined below. Let u denote the capacities allocated in the current solution. The operation $\text{swap}(u^* - u)$ adjusts the capacity of each facility i from u_i to u_i^* and reroutes excess demand from facilities with $u_i > u_i^*$ to facilities with $u_i^* > u_i$ which after adjustment have excess free capacity. The cost of this operation is equal to the total facility cost at capacities u^* ($c_f(S^*)$) minus the total facility cost at capacities u ($c_f(S)$) plus the *rerouting cost* $\sum_{i,j} c_{ij} \delta_{ij}$, where δ_{ij} denotes the amount of flow rerouted from facility i to facility j .

The plan for the rest of this section is as follows: We first show how to reroute the demand in the operation $\text{swap}(u^* - u)$ so that the rerouting cost is small. Then we show how to replace this fictitious **swap** operation with a list of **pivot** operations and bound the total cost of these operations in terms of the global **swap** operation. Finally, if S is locally optimal, we know that the cost of each **pivot** operation must be greater than or equal to zero. Summing these inequalities over all **pivot** operations in our list gives us a bound on the facility cost of S .

The exchange graph. Without loss of generality we shall assume that the allocated capacity at each facility is equal to the demand served by it in both S and S^* : $u_i = \sum_j x_{ij}$ and $u_i^* = \sum_j x_{ij}^*$. Let $\delta_i = u_i - u_i^*$ be the amount of excess or deficit of demand of each facility in our fictitious **swap** operation; note that $\sum_i \delta_i = 0$. We have not yet specified how we reassign demand. To do this, we set up a transshipment problem on the graph G : facilities $U = \{i \in F \mid \delta_i > 0\}$ are the sources, and facilities $U^* = \{i \in F \mid \delta_i < 0\}$ are the sinks. The goal is to find a flow of small cost such that exactly δ_s units of flow emanate from every source s and $-\delta_t$ units enter into each sink t . The cost of shipping a unit of flow between two vertices s and t is equal to their distance c_{st} , and there are no capacities on the edges.

Note that each flow that is feasible for the transshipment problem immediately gives a way of reassigning demand in the imaginary **swap** operation, and the cost of the flow is equal to the rerouting cost of the **swap** operation. We claim that there is a solution to the transshipment problem with low cost. We refer the reader to the full paper for a proof of the following lemma.

Lemma 2. *The transshipment problem defined above has a solution of cost at most $c_s(S) + c_s(S^*)$.*

We do not know how to find a good **swap** operation efficiently. Thus, we cannot design an efficient algorithm based on this operation. However, we would

like to illustrate the style of argument we will be using with the following simple lemma.

Lemma 3. *Let S be a solution that can not be improved by any swap or add operation. Then $c_f(S) \leq 2c(S^*)$.*

Proof. The facility cost of the **swap**($u^* - u$) operation is $\sum_{i \in F} f_i(u_i^*) - f_i(u_i) = c_f(S^*) - c_f(S)$, while the rerouting cost by Lemma 2 is at most $c_s(S) + c_s(S^*)$. Since S is locally optimal with respect to swaps, the cost of the swap must be nonnegative: $0 \leq -c_f(S) + c_f(S^*) + c_s(S^*) + c_s(S)$. By Lemma 1, we have $c_s(S) \leq c(S^*)$. Plugging this into the above inequality and rearranging we get the claimed bound. $\square$

The exchange forest. Let us consider a flow y that is an optimal solution to the transshipment problem defined above. We claim that without loss of generality, the set of edges with nonzero flow forms a forest. Indeed, suppose that there is a cycle of nonzero edges. Augmenting the flow along the cycle in either direction must not increase the cost of the flow (otherwise augmenting in the opposite direction would decrease it, contradicting the optimality of y). Hence we can keep augmenting until the flow on one of the edges becomes zero; this way we can remove all cycles one by one. Similarly we can assume that there is no flow between a pair of sources (facilities with $\delta_i > 0$) or sinks (facilities with $\delta_i < 0$). If there was a positive flow from a source s_1 to another source s_2 , it must be the case that it eventually arrives at some sink (or sinks) t . By triangle inequality, the flow can be sent directly from s_1 to t without increasing the cost.

Hence, the flow y can be thought of as a collection of bipartite trees, with edges leading between alternating layers of vertices from U^* and U . We root each tree at an arbitrary facility in U^* .

A list of pivot operations. We are ready to proceed with our plan and decompose the imaginary **swap** operation into a list of local **pivot** operations. Each of these **pivot** operations decreases the capacity of some facilities from u_i to u_i^* (we say that such a facility is *closed*) and increases some others from u_i to at most u_i^* (we say that such a facility is *opened*). Note that only facilities in U^* can be opened, and only facilities in U can be closed by the operations in the decomposition. Also, for each **pivot** operation, we specify a way to reroute the demand from the facilities that are closed to facilities that are open *along the edges of the exchange forest*.

Therefore, since f_i 's are non-decreasing, the cost of a pivot operation in our list is at most

$$\sum_{i \in A} (f_i(u_i^*) - f_i(u_i)) + \sum_e c_e \delta_e, \quad (1)$$

where A is the set of facilities affected (i.e., either opened or closed) by the operation, and δ_e is the amount of demand rerouted through edge e of the exchange forest.

We choose the list of operations in such a way that they satisfy the following three properties.

1. Each facility in U is closed by exactly one operation.

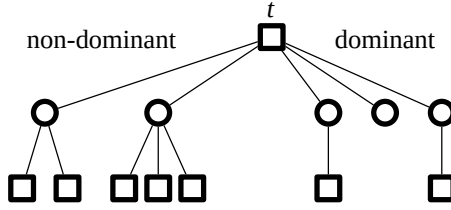


Fig. 1. The subtree $\mathcal{T}_t$. Circles denote facilities from U , squares facilities from U^* .

2. Each facility in U^* is opened by at most 4 operations.
3. For every edge e of the exchange graph, the total amount of demand rerouted through e is at most 3 times the amount of flow y along e .

The following lemma shows that finding a list of **pivot** operations with the above properties is enough for bounding the facility cost of S .

Lemma 4. *Assume there is a list of **pivot** operations satisfying the above properties. Then, the facility cost of the solution S is at most $c_f(S) \leq 4 \cdot c_f(S^*) + 3 \cdot (c_s(S) + c_s(S^*))$.*

Proof. Since S is a local optimum, the cost of every local operation, and therefore the upper bound given in (1) for the cost of each operation in our list must be non-negative. Adding up these inequalities, and using the above properties and the definition of U and U^* , we get $4 \sum_{i \in U^*} (f_i(u_i^*) - f_i(u_i)) + 3 \sum_{s,t} c_{st} y_{st} \geq \sum_{i \in U} (f_i(u_i) - f_i(u_i^*))$, where $\sum_{s,t} c_{st} y_{st}$ is the cost of flow y , which is at most $c_s(S) + c_s(S^*)$ by Lemma 2. Adding $\sum_{i \in U^*} f_i(u_i) + \sum_{i \in U} f_i(u_i^*)$ to both sides and rearranging, we get $c_f(S) \leq c_f(S^*) + 3 \sum_{i \in U^*} (f_i(u_i^*) - f_i(u_i)) + 3 \sum_{s,t} c_{st} y_{st} \leq 4 \cdot c_f(S^*) + 3 \cdot (c_s(S) + c_s(S^*))$. $\square$

In the rest of this section, we will present a list of operations satisfying the above properties.

Decomposing the trees. Recall that the edges on which the flow y is nonzero form a forest. Root each tree $\mathcal{T}$ at some facility $r \in U^*$. For a vertex $t \in U^*$, define $C(t)$ to be the set of children of t . Since the flow y is bipartite, the children $C(t) \subseteq U$. For a $t \in U^*$ that is not a leaf of $\mathcal{T}$, let $\mathcal{T}_t$ be the subtree of depth at most 2 rooted at t containing all children and grandchildren of t . (See Figure 1).

We define a set of operations for every such subtree $\mathcal{T}_t$. In case t has no grandchildren, we consider a single **pivot**(t, Δ) operation that has t as its pivot, opens t , closes the children $C(t)$, and reroutes the traffic from facilities in $C(t)$ to t through the edges of the exchange graph. This operation is feasible, as the total capacity closed is less than or equal to the capacity opened (i.e., $u_t^* - u_t$). Also, in this case the reassignment cost along each edge in $\mathcal{T}_t$ is bounded by the cost of the flow on that edge.

We now consider the general case in which the tree $\mathcal{T}_t$ has depth 2. We divide the children of t into two sets. A node $s \in C(t)$ is *dominant*, if at least half of the total flow emanating from s goes to t (i.e. $y_{st} \geq \frac{1}{2} \sum_{t' \in U} y_{st'}$) and

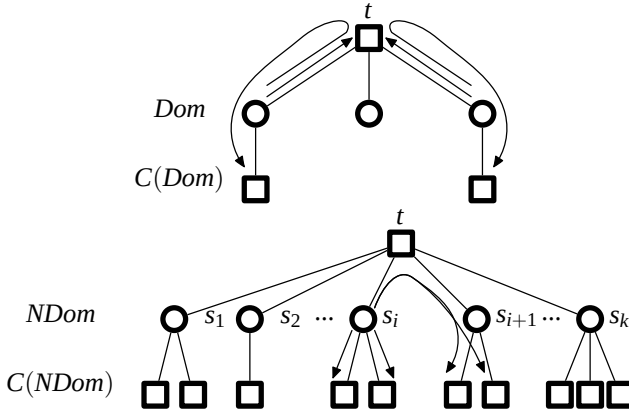


Fig. 2. Reassigning demand. The operation $\text{pivot}(t, \Delta)$ shown on the left closes all facilities in Dom . The pivot operation on the right closes a facility $s_i \in Dom$.

non-dominant otherwise. Let Dom and $NDom$ be the set of dominant and non-dominant facilities respectively. (See Figure 1).

We close all dominant nodes in a single operation. We let t to be the pivot point, close all facilities in Dom and open t and all children of Dom . Note that since we decided to open all neighbors of Dom , there will be enough free capacity to accommodate the excess demand. Figure 2 (left panel) shows how the demand is rerouted.

We can not afford to do the same with the non-dominant nodes, because the pivot operation requires that all the affected demands are first shipped to the pivot. Since the flow from a node $s \in C(t)$ to its children may be arbitrarily larger than the flow from s to t , the cost of shipping it to t and then back to the children of s might be prohibitively large. Instead, we use a separate pivot operation for each $s \in C(t)$ that closes s and opens all children of s . The operation may not be feasible, because we still may have to deal with the leftover demand that s was sending to t . The opening cost of t may be large, hence we need to avoid opening t in a pivot operation for every $s \in NDom$.

We order the elements $s_1, s_2, \dots, s_k$ of $NDom$ by the amount of flow they send to the root t , i.e., $y_{s_1,t} \leq y_{s_2,t} \leq \dots \leq y_{s_k,t}$. For s_i , $1 \leq i < k$, we consider the operation with s_i as the pivot, that closes s_i and opens the set $C(s_i) \cup C(s_{i+1})$. By our ordering, the amount of flow from s_i to t is no more than the amount of flow from s_{i+1} to t , and since s_{i+1} is non-dominant, this is also less than or equal to the total capacity opened in $C(s_{i+1})$. Hence the opened capacity at $C(s_i) \cup C(s_{i+1})$ is enough to cover the excess demand arising from closing capacity at s_i . For the last facility s_k we consider the operation with s_k as a pivot, that closes s_k and opens facility t as well as facilities in $C(s_k)$. (See Figure 2).

We have defined a set of pivoting operations for each tree $\mathcal{T}_t$ for $t \in U^*$. To finish the analysis, one would need to verify that these operations satisfy the

properties 1–3 above. These properties, together with Lemmas 1 and 4 imply the following.

Theorem 3. *Consider a UniFL instance. Let S^* be an optimal solution for this instance, and S be a locally optimal solution with respect to the add and pivot operations. Then $c_s(S) \leq c_s(S^*) + c_f(S^*)$ and $c_f(S) \leq 6c_s(S^*) + 7c_f(S^*)$.*

4.3 Establishing Polynomial Running Time

As discussed in Section 3, in order to be able to guarantee a polynomial running time, we need to make a significant improvement in every step. Therefore, at the end, we will find a solution S such that no local operation can improve its cost by more than a factor of $\frac{\varepsilon}{5n}c(S)$, whereas in our analysis we assumed the S cannot be improved by local operations at all.

The proof of Lemma 1 as well as Lemma 4 is based on selecting a set of at most n operations and summing up the corresponding inequalities saying that the cost of each operation is nonnegative. By relaxing local optimality we must add up to n times the term $\frac{\varepsilon}{5n}c(S)$ to the resulting bound; that is, the claim of Lemma 1 becomes $c_s(S) \leq c_f(S^*) + c_s(S^*) + \frac{\varepsilon}{5}c(S)$ and the bound in Lemma 4 changes to $c_f(S) \leq 3(c_s(S^*) + c_s(S)) + 4c_f(S^*) + \frac{\varepsilon}{5}c(S)$. Combining these inequalities we get $c(S) \leq 8c(S^*) + \varepsilon c(S)$.

Theorem 4. *The algorithm described in Section 3 terminates after at most $\frac{10n}{\varepsilon} \ln \frac{c(S)}{c(S^*)}$ iterations and outputs a solution of cost at most $8/(1 - \varepsilon)$ times the optimum.*

Using standard scaling techniques, it is possible to improve the approximation ratio of our algorithm to $\sqrt{15} + 4 \approx 7.873$. See the full version for details.

5 Conclusion

In this paper, we presented the first constant-factor approximation algorithm for the universal facility location problem, generalizing many of the previous results on various facility location problems and slightly improving the best approximation factor known for the hard-capacitated facility location problem. We proved that the approximation factor of our algorithm is at most $8 + \varepsilon$, however, we do not know of any example on which our algorithm performs worse than 4 times the optimum (the example given in [18] shows that our algorithm sometimes performs 4 times worse than the optimum). A tight analysis of our algorithm remains an open question. Furthermore, the only inapproximability lower bound we know for the universal facility location problem is a bound of 1.463 proved by Guha and Khuller [8] for the uncapacitated facility location problem. Finding a better lower bound for the more general case of universal facility location is an interesting open question. In the non-metric case (i.e., when connection costs do not obey the triangle inequality), the UniFL problem (even in the uncapacitated case) is hard to approximate within a factor better than

$O(\log n)$. Another open question is to find a $O(\log n)$ -approximation algorithm for the non-metric UniFL problem.

Our algorithm was based on the assumption that the facility cost functions are non-decreasing. While this assumption holds in most practical cases, there are cases where solving the universal facility location problem with decreasing cost functions can be helpful. The load balanced (a.k.a. lower bounded) facility location problem is an example. This problem, which is a special case of UniFL with $f_i(u) = \bar{f}_i \cdot [u > 0] + \infty \cdot [u < \bar{l}_i]$ for given $\bar{f}_i$ and $\bar{l}_i$, was first defined by Karger and Minkoff [11] and Guha et al. [9] and used to solve other location problems. We still do not know of any constant factor approximation algorithm for this problem, and more generally for UniFL with decreasing cost functions. As in the hard-capacitated facility location problem, the integrality gap of the natural LP relaxation of this problem is unbounded, and therefore local search seems to be a natural approach. However, our analysis does not work for this case, since **add** operations are not necessarily feasible, and therefore Lemma 1 does not work.

References

1. V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. In *Proceedings of 33rd ACM Symposium on Theory of Computing*, 2001.
2. M.L. Balinski. On finding integer solutions to linear programs. In *Proc. IBM Scientific Computing Symposium on Combinatorial Problems*, pages 225–248, 1966.
3. P. Bauer and R. Enders. A capacitated facility location problem with integer decision variables. In *International Symposium on Math. Programming*, 1997.
4. M. Charikar and S. Guha. Improved combinatorial algorithms for facility location and k-median problems. In *Proceedings of FOCS'99*, pages 378–388, 1999.
5. N. Christofides and J.E. Beasley. An algorithm for the capacitated warehouse location problem. *European Journal of Operational Research*, 12:19–28, 1983.
6. F.A. Chudak and D.P. Williamson. Improved approximation algorithms for capacitated facility location problems. In *Integer Programming and Combinatorial Optimization (Graz, 1999)*, volume 1610 of *Lecture Notes in Computer Science*, pages 99–113. Springer, Berlin, 1999.
7. E. Feldman, F.A. Lehrer, and T.L. Ray. Warehouse locations under continuous economies of scale. *Management Science*, 12:670–684, 1966.
8. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31:228–248, 1999.
9. S. Guha, A. Meyerson, and K. Munagala. Hierarchical placement and network design problems. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000.
10. M. Hajiaghayi, M. Mahdian, and V.S. Mirrokni. The facility location problem with general cost functions. *Networks*, 42(1):42–47, August 2003.
11. D. Karger and M. Minkoff. Building Steiner trees with incomplete global knowledge. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000.
12. L. Kaufman, M.V. Eede, and P. Hansen. A plant and warehouse location problem. *Operations Research Quarterly*, 28:547–554, 1977.

13. M.R. Korupolu, C.G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, January 1998.
14. A.A. Kuehn and M.J. Hamburger. A heuristic program for locating warehouses. *Management Science*, 9:643–666, 1963.
15. M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2002)*, 2002.
16. M. Mahdian, Y. Ye, and J. Zhang. A 2-approximation algorithm for the soft-capacitated facility location problem. to appear in APPROX 2003, 2003.
17. R.M. Nauss. An improved algorithm for the capacitated facility location problem. *Journal of Operational Research Society*, 29:1195–1202, 1978.
18. M. Pál, E. Tardos, and T. Wexler. Facility location with hard capacities. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, 2001.
19. D.B. Shmoys. Approximation algorithms for facility location problems. In K. Jansen and S. Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, volume 1913 of *Lecture Notes in Computer Science*, pages 27–33. Springer, Berlin, 2000.
20. J.F. Stollsteimer. A working model for plant numbers and locations. *J. Farm Econom.*, 45:631–645, 1963.

A Method for Creating Near-Optimal Instances of a Certified Write-All Algorithm (Extended Abstract)

Grzegorz Malewicz*

Laboratory for Computer Science, Massachusetts Institute of Technology
200 Technology Square, NE43-205, Cambridge, MA 02139
malewicz@theory.lcs.mit.edu

Abstract. This paper shows how to create near-optimal instances of the Certified Write-All algorithm called AWT that was introduced by Anderson and Woll [2]. This algorithm is the best known deterministic algorithm that can be used to simulate n synchronous parallel processors on n asynchronous processors. In this algorithm n processors update n memory cells and then signal the completion of the updates. The algorithm is instantiated with q permutations, where q can be chosen from a wide range of values. When implementing a simulation on a specific parallel system with n processors, one would like to use an instance of the algorithm with the best possible value of q , in order to maximize the efficiency of the simulation. This paper shows that the choice of q is critical for obtaining an instance of the AWT algorithm with near-optimal work. For any $\epsilon > 0$, and any large enough n , work of any instance of the algorithm must be at least $n^{1+(1-\epsilon)\sqrt{2\ln\ln n/\ln n}}$. Under certain conditions, however, that q is about $e^{\sqrt{1/2\ln n \ln \ln n}}$ and for infinitely many large enough n , this lower bound can be nearly attained by instances of the algorithm with work at most $n^{1+(1+\epsilon)\sqrt{2\ln\ln n/\ln n}}$. The paper also shows a penalty for not selecting q well. When q is significantly away from $e^{\sqrt{1/2\ln n \ln \ln n}}$, then work of any instance of the algorithm with this displaced q must be considerably higher than otherwise.

1 Introduction

This paper shows how to create near-optimal instances of the Certified Write-All algorithm called AWT that was introduced by Anderson and Woll [2]. In this algorithm n processors update n memory cells and then signal the completion of the updates. The algorithm is instantiated with q permutations, where q can

* The work of Grzegorz Malewicz was done during a visit to the Supercomputing Technologies Group (“the Cilk Group”), Massachusetts Institute of Technology, headed by Prof. Charles E. Leiserson. Grzegorz Malewicz was visiting this group during the 2002/2003 academic year while in his final year of the Ph.D. program at the University of Connecticut, where his advisor is Prof. Alex Shvartsman. Part of this work was supported by the Singapore/MIT Alliance.

be chosen from a wide range of values. This paper shows that the choice of q is critical for obtaining an instance of the AWT algorithm with near-optimal work.

Many existing parallel systems are asynchronous. However, writing correct parallel programs on an asynchronous shared memory system is often difficult, for example because of data races, which are difficult to detect in general. When the instructions of a parallel program are written with the intention of being executed on a system that is synchronous, then it is easier for a programmer to write correct programs, because it is easier to reason about synchronous parallel programs than asynchronous ones. Therefore, in order to improve productivity in parallel computing, one could offer programmers illusion that their programs run on a parallel system that is synchronous, while in fact the programs would be simulated on an asynchronous system.

Simulations of a parallel system that is synchronous on a system that is asynchronous have been studied for over a decade now (see e.g., [8,9]). Simplifying considerably, simulations assume that there is a system with p asynchronous processors, and the system is to simulate a program written for n synchronous processors. The simulations use three main ideas: idempotence, load balancing, and synchronization. Specifically, the execution of the program is divided into a sequence of phases. A phase executes an instruction of each of the n synchronous programs. The simulation executes a phase in two stages: first the n instructions are executed and the results are saved to a scratch memory, only then cells of the scratch memory are copied back to desired cells of the main memory. This ensures that the result of the phase is the same even if multiple processors execute the same instruction in a phase, which may happen due to asynchrony. The p processors run a load balancing algorithm to ensure that the n instructions of the phase are executed quickly despite possibly varying speeds of the p processors. In addition, the p processors should be synchronized at every stage, so as to ensure that the simulated program proceeds in lock-step.

One challenge in realizing the simulations is the problem of “late writers” i.e., when a slow processor clobbers the memory of a simulation with a value from an old phase. This problem has been addressed in various ways (see e.g., [3,13]). Another challenge is the development of efficient load-balancing and synchronization algorithms. This challenge is abstracted as the Certified Write-All (CWA) problem. In this problem, introduced in a slightly different form by Kanellakis and Shvartsman [7], there are p processors, an array w with n cells and a flag f , all initially 0, and the processors must set the n cells of w to 1, and then set f to 1. A simulation uses an algorithm that solves the CWA problem, and the overhead of the simulation depends on efficiency of the algorithm. The efficiency of the algorithm is measured by *work* that is equal to the worst-case total number of instructions executed by the algorithm. Hence it is desirable to develop low-work algorithms that solve the CWA problem.

Deterministic algorithms that solve the CWA problem on an asynchronous system can be used to create simulations that have bounded worst-case overhead. Thus several deterministic algorithms have been studied [2,4,5,6,8,14]. The class of algorithms for the case when $p = n$ is especially interesting because they have

high parallelism. When such algorithm is used in a simulation, the simulation of a given synchronous program for $p = n$ processors may be faster, as compared to the simulation that uses an algorithm for $p \ll n$ processors, simply because in the former case more processors are available to simulate the program. However, the potential of producing a faster simulation can only be realized when the algorithm used has low work, so that not much computing resources are wasted during any simulation phase.

The best deterministic algorithm that solves the CWA problem on an asynchronous system for the case when $p = n$ was introduced by Anderson and Woll [2]. This algorithm is called AWT, and it generalizes the algorithm X of Buss et al. [4]. The AWT algorithm is instantiated with a list of q permutations on $\{1, \dots, q\}$. Anderson and Woll showed that for any $\epsilon > 0$, there is q , a list of q permutations with desired *contention*, and a constant c_q , such that for any $h > 0$, the algorithm for $p = q^h$ processors and $n = p$ cells instantiated with the list, has work at most $c_q \cdot n^{1+\epsilon}$. Note that this upper bound includes a multiplicative constant factor that is a function of q . While the result that an $O(n^{1+\epsilon})$ work algorithm can be found is very interesting, a different search objective will occur when a simulation is developed for a specific parallel system.

A specific parallel system will have a fixed number p of processors. It is possible to create many instances of the AWT algorithm for these p processors and $n = p$ cells, that differ by the number q of permutations used to create an instance. It is possible that work of these different instances is different. If this is indeed the case, then it is interesting to find an instance with the lowest work, so as to create a relatively more efficient simulation on this parallel system.

Contributions. This paper shows how to create near-optimal instances of the AWT algorithm of Anderson and Woll. In this algorithm p processors update $n = p$ memory cells and then signal the completion of the updates. The algorithm is instantiated with q permutations on $\{1, \dots, q\}$, where q can be chosen from a wide range of values. This paper shows that the choice of q is critical for obtaining an instance of the AWT algorithm with near-optimal work. Specifically, we show a tight (up to an absolute constant) lower bound on work of the AWT algorithm instantiated with a list of q permutations (appearing in Lemma 4). This lower bound generalizes the Lemma 5.20 of Anderson and Woll by exposing a constant that depends on q and on the contention of the list. We then combine our lower bound with a lower bound on contention of permutations given by Lovász [11] and Knuth [10], to show that for any $\epsilon > 0$, work of any instance must be at least $n^{1+(1-\epsilon)\sqrt{2 \ln \ln n / \ln n}}$, for any large enough n (appearing in Theorem 1). The resulting bound is nearly optimal, as demonstrated by our method for creating instances of the AWT algorithm. We show that for any $\epsilon > 0$ and for any m that is large enough, when $q = \lceil e^{\sqrt{1/2 \ln m \ln \ln m}} \rceil$, and $h = \lceil \sqrt{2 \ln m / \ln \ln m} \rceil$, then there exists an instance of the AWT algorithm for $p = q^h$ processors and $n = p$ cells that has work at most $n^{1+(1+\epsilon)\sqrt{2 \ln \ln n / \ln n}}$ (appearing in Theorem 2). We also prove that there is a penalty if one selects a q that is too far away from $e^{\sqrt{1/2 \ln n \ln \ln n}}$. For any fixed $r \geq 2$, and any large enough n , work is at

least $n^{1+r/3 \cdot \sqrt{2 \ln \ln n / \ln n}}$, whenever the AWT algorithm is instantiated with q permutations, such that $16 \leq q \leq e^{\sqrt{1/2 \ln n \ln \ln n / (r \cdot \ln \ln n)}}$ or $e^{r \cdot \sqrt{1/2 \ln n \ln \ln n}} \leq q \leq n$ (appearing in Proposition 1).

Paper organization. The remainder of the paper is organized as follows. In Section 2, we report on some existing results on contention of permutations and present the AWT algorithm of Anderson and Woll. In Section 3, we show our optimization argument that leads to the development of a method for creating near-optimal instances of the AWT algorithm. Finally, in Section 4, we conclude with future work. Due to lack of space some proofs were omitted, and they will appear the upcoming doctoral dissertation of the author.

2 Preliminaries

For a permutation ρ on $[q] = \{1, \dots, q\}$, $\rho(v)$ is a *left-to-right maximum* [10] if it is larger than all of its predecessors i.e., $\rho(v) > \rho(1), \rho(v) > \rho(2), \dots, \rho(v) > \rho(v-1)$. The *contention* [2] of ρ with respect to a permutation α on $[q]$, denoted as $Cont(\rho, \alpha)$, is defined as the number of left-to-right maxima in the permutation $\alpha^{-1}\rho$ that is a composition of α^{-1} with ρ . For a list $R_q = \langle \rho_1, \dots, \rho_q \rangle$ of q permutations on $[q]$ and a permutation α on $[q]$, the contention of R_q with respect to α is defined as $Cont(R_q, \alpha) = \sum_{v=1}^q Cont(\rho_v, \alpha)$. The contention of the list of permutations R_q is defined as $Cont(R_q) = \max_{\alpha \text{ on } [q]} Cont(R_q, \alpha)$.

Lovász [11] and Knuth [10] showed that the expectation of the number of left-to-right maxima in a random permutation on $[q]$ is H_q (H_q is the q th harmonic number). This immediately implies the following lower bound on contention of a list of q permutations on $[q]$.

Lemma 1. [11,10] *For any list R_q of q permutations on $[q]$, $Cont(R_q) \geq qH_q > q \ln q$.*

Anderson and Woll [2] showed that for any q there is a list of q permutations with contention $3qH_q$. Since $H_q / \ln q$ tends to 1, as q tends to infinity, the following lemma holds.

Lemma 2. [2] *For any q that is large enough, there exists a list of q permutations on $[q]$ with contention at most $4 \cdot q \ln q$.*

We describe the algorithm AWT of Anderson and Woll [2] that solves the CWA problem when $p = n$. There are $p = q^h$ processors, $h \geq 1$, and the array w has $n = p$ cells. The identifier of a processor is represented by a distinct string of length h over the alphabet $[q]$. The algorithm is instantiated with a list of q permutations $R_q = \langle \rho_1, \dots, \rho_q \rangle$ on $[q]$, and we write $AWT(R_q)$ when we refer to the instance of algorithm AWT for a given list of permutations R_q . This list is available to every processor (in its local memory). Processors have access to a shared q -ary tree called *progress tree*. Each node of the tree is labeled with a string over alphabet $[q]$. Specifically, a string $s \in [q]^*$ that labels a node identifies the path from the root to the node (e.g., the root is labeled with the

AWT(R_q)	
01	$Traverse(h, \lambda)$
02	set f to 1 and Halt
$Traverse(i, s)$	
01	if $i = 0$ then
02	$w[val(s)] := 1$
03	else
04	$j := q_i$
05	for $v := 1$ to q
06	$a := \rho_j(v)$
07	if $b_{s \cdot a} = 0$ then
08	$Traverse(i - 1, s \cdot a)$
09	$b_{s \cdot a} := 1$

Fig. 1. The instance $AWT(R_q)$ of an algorithm of Anderson and Woll, as executed by a processor with identifier $\langle q_1 \dots q_h \rangle$. The algorithm uses a list of q permutations $R_q = \langle \rho_1, \dots, \rho_q \rangle$.

empty string λ , the leftmost child of the root is labeled with the string 1). For convenience, we say node s , when we mean the node labeled with a string s . Each node s of the tree, apart from the root, contains a *completion bit*, denoted by b_s , initially set to 0. Any leaf node s is canonically assigned a distinct number $val(s) \in \{0, \dots, n - 1\}$.

The algorithm, shown in Figure 1, starts by each processor calling procedure $AWT(R_q)$. Each processor traverses the q -ary progress tree by calling a recursive procedure $Traverse(h, \lambda)$. When a processor visits a node that is the root of a subtree of height i (the root of the progress tree has height h) the processor takes the i th letter j of its identifier (line 04) and attempts to visit the children in the order established by the permutation ρ_j . The visit to a child $a \in [q]$ *succeeds* only if the completion bit $b_{s \cdot a}$ for this child is still 0 at the time of the attempt (line 07). In such case, the processor recursively traverses the child subtree (line 08), and later sets to one the completion bit of the child node (line 09). When a processor visits a leaf s , the processor performs an assignment of 1 to the cell $val(s)$ of the array w . After a processor has finished the recursive traversal of the progress tree, the processor sets f to 1 and halts.

We give a technical lemma that will be used to solve a recursive equation in the following section.

Lemma 3. *Let h and q be integers, $h \geq 1$, $q \geq 2$, and $k_1 + \dots + k_q = c > 0$. Consider a recursive equation $W(0, r) = r$, and $W(i, r) = r \cdot q + \sum_{v=1}^q W(i - 1, k_v \cdot r/q)$, when $i > 0$. Then for any r ,*

$$W(h, r) = r \left(q \cdot \frac{(c/q)^h - 1}{c/q - 1} + (c/q)^h \right).$$

3 Near-Optimal Instances of AWT

This section presents a method for creating near-optimal instances of the AWT algorithm of Anderson and Woll. The main idea of this section is that for fixed number p of processors and $n = p$ cells of the array w , work of an instance

of the AWT algorithm depends on the number of permutations used by the instance, along with their contention. This observation has several consequences. It turns out (not surprisingly) that work increases when contention increases, and conversely it becomes the lowest when contention is the lowest. Here a lower bound on contention of permutations given by Lovász [11] and Knuth [10] is very useful, because we can bound work of any instance from below, by an expression in which the value of contention of the list used in the instance is replaced with the value of the lower bound on contention. Then we study how the resulting lower bound on work depends on the number q of permutations on $[q]$ used by the instance. It turns out that there is a single value for q , where the bound attains the global minimum. Consequently, we obtain a lower bound on work that, for fixed n , is independent of both the number of permutations used and their contention. Our bound is near-optimal. We show that if we instantiate the AWT algorithm with about $e^{\sqrt{1/2 \ln n \ln \ln n}}$ permutations that have small enough contention, then work of the instance nearly matches the lower bound. Such permutations exist as shown by Anderson and Woll [2]. We also show that when we instantiate the AWT algorithm with much fewer or much more permutations, then work of the instance must be significantly greater than the work that can be achieved. Details of the overview follow.

We will present a tight bound on work of any instance of the AWT algorithm. Our lower bound generalizes the Lemma 5.20 of Anderson and Woll [1]. The bound has an explicit constant which was hidden in the analysis given in the Lemma 5.20. The constant will play a paramount role in the analysis presented in the remainder of the section.

Lemma 4. *Work W of the AWT algorithm for $p = q^h$ processors, $h \geq 1$, $q \geq 2$, and $n = p$ cells, instantiated with a list $R_q = \langle \rho_1, \dots, \rho_q \rangle$ of q permutations on $[q]$, is bounded by $\frac{c}{84} \cdot n^{1+\log_q \frac{\text{Cont}(R_q)}{q}} \leq W \leq c \cdot n^{1+\log_q \frac{\text{Cont}(R_q)}{q}}$, where $c = \frac{28q^2}{\text{Cont}(R_q)}$.*

Proof. The idea of the lemma is to carefully account for work spent on traversing the progress tree, and spent on writing to the array w . The lower bound will be shown by designing an execution during which the processors will traverse the progress tree in a specific, regular manner. This regularity will allow us to conveniently bound from below work inside a subtree, by work done at the root of the subtree and work done by quite large number of processors that traverse the child subtrees in a regular manner. A similar recursive argument will be used to derive the upper bound.

Consider any execution of the algorithm. We say that the execution is *regular* at a node s (recall that s is a string from $[q]^*$) iff the following three conditions hold:

- (i) the r processors that ever visit the node during the execution, visit the node at the same time,
- (ii) at that time, the completion bit of any node of the subtree of height i rooted at the node s is equal to 0,

- (iii) if a processor visits the node s , and x is the suffix of length $h - i$ of the identifier of the processor, then the q^i processors that have x as a suffix of their identifiers, also visit the node during the execution.

We define $W(i, r)$ to be the largest number of basic actions that r processors perform inside a subtree of height i , from the moment when they visit a node s that is the root of the subtree until the moment when each of the visitors finishes traversing the subtree, maximized across the executions that are regular at s and during which exactly r processors visit s (if there is no such execution, we put $-\infty$). Note that the value of $W(i, r)$ is well-defined, as it is independent of the choice of a subtree of height i (any pattern of traversals that maximizes the number of basic actions performed inside a subtree, can be applied to any other subtree of the same height), and of the choice of the r visitors (suffixes of length $h - i$ do not affect traversal within the subtree). There exists an execution that is regular at the root of the progress tree, and so the value of $W(h, n)$ bounds work of $\text{AWT}(R_q)$ from below.

We will show a recursive formula that bounds $W(i, r)$ from below. We do it by designing an execution recursively. The execution will be regular at every node of the progress tree. We start by letting the q^h processors visit the root at the same time. For the recursive step, assume that the execution is regular at a node s that is the root of a subtree of height i , and that exactly r processors visit the node. We first consider the case when s is an internal node i.e., when $i > 0$. Based on the i -th letter of its identifier, each processor picks a permutation that gives the order in which completion bits of the child nodes will be read by the processor. Due to regularity, the r processors can be partitioned into q collections of equal cardinality, such that for any collection j , each processor in the collection checks the completion bits in the order given by ρ_j . Let for any collection, the processors in the collection check the bits of the children of the node in lock step (the collection behaves as a single “virtual” processor). Then, by Lemma 2.1 of Anderson and Woll [2], there is a pattern of delays so that every processor in some $k_v \geq 1$ collections succeeds in visiting the child $s \cdot v$ of the node at the same time. Thus the execution is regular at any child node. The lemma also guarantees that $k_1 + \dots + k_q = \text{Cont}(R_q)$, and that these $k_1, \dots, k_q$ do not depend on the choice of the node s . Since each processor checks q completion bits of the q children of the node, the processor executes at least q basic actions while traversing the node. Therefore, $W(i, r) \geq rq + \sum_{v=1}^q W(i - 1, k_v \cdot r/q)$, for $i > 0$. Finally, suppose that s is a leaf i.e., that $i = 0$. Then we let the r processors work in lock step, and so $W(0, r) \geq r$.

We can bound the value of $W(h, n)$ using Lemma 3, the fact that $h = \log_q n$, and that for any positive real a , $a^{\log_q n} = n^{\log_q a}$, as follows

$$\begin{aligned} W(h, n) &\geq n \cdot (\text{Cont}(R_q)/q)^h \left(q \cdot \frac{1 - (q/\text{Cont}(R_q))^h}{\text{Cont}(R_q)/q - 1} + 1 \right) \\ &= n^{1+\log_q(\text{Cont}(R_q)/q)} \left(q^2/\text{Cont}(R_q) \cdot \frac{1 - (q/\text{Cont}(R_q))^h}{1 - q/\text{Cont}(R_q)} + 1 \right) \end{aligned}$$

$$\begin{aligned}
&> q^2 / \text{Cont}(R_q) \cdot n^{1+\log_q(\text{Cont}(R_q)/q)} \left(1 - (q/\text{Cont}(R_q))^h\right) \\
&\geq 1/3 \cdot q^2 / \text{Cont}(R_q) \cdot n^{1+\log_q(\text{Cont}(R_q)/q)},
\end{aligned}$$

where the last inequality holds because for all $q \geq 2$, $q/\text{Cont}(R_q) \leq 2/3$, and $h \geq 1$.

The argument for proving an upper bound is similar to the above argument for proving the lower bound. The main conceptual difference is that processors may write completion bits in different order for different internal nodes of the progress tree. Therefore, while the coefficients $k_1, \dots, k_q$ were the same for each node during the analysis above, in the analysis of the upper bound, each internal node s has its own coefficients $k_1^s, \dots, k_q^s$ that may be different for different nodes. The proof of the upper bound is omitted.

How does the bound from the lemma above depend on contention of the list R_q ? We should answer this question so that when we instantiate the AWT algorithm, we know whether to choose permutations with low contention or perhaps with high contention. The answer to the question may be not so clear at first, because for any given q , when we take a list R_q with lower contention, then although the exponent of n is lower, but the constant c is higher. In the lemma below we study this tradeoff, and demonstrate that it is indeed of advantage to choose lists of permutations with as small contention as possible.

Lemma 5. *The function $c \mapsto q^2/c \cdot n^{\log_q c}$, where $c > 0$ and $n \geq q \geq 2$, is a non-decreasing function of c .*

The above lemma, simple as it is, is actually quite useful. In several parts of the paper we use a list of permutations, for which we only know an upper bound or a lower bound on contention. This lemma allows us to bound work respectively from above or from below, even though we do not actually know the exact value of contention of the list.

We would like to find out how the lower bound on work depends on the choice of q . The subsequent argument shows that careful choice of the value of q is essential, in order to guarantee low work. We begin with two technical lemmas, the second of which bounds from below the value of a function occurring in Lemma 4.

The lemma below shows that an expression that is a function of x must vanish inside a “slim” interval. The key idea of the proof of the lemma is that x^2 creates in the expression a highest order summand with factor either $1/2$ or $(1 + \epsilon)/2$ depending on which of the two values of x we take, while $\ln x$ creates a summand of the same order with factor $1/2$ independent of the value of x . As a result, for the first value of x , the former “is less positive” than the later “is negative”, while when x has the other value, then the former “is more positive” than the later “is negative”. The proof is omitted.

Lemma 6. *Let $\epsilon > 0$ be any fixed constant. Then for any large enough n , the expression $x^2 - x + (1 - \ln x) \cdot \ln n$ is negative when $x = x_1 = \sqrt{1/2 \ln n \ln \ln n}$, and positive when $x = x_2 = \sqrt{(1 + \epsilon)/2 \ln n \ln \ln n}$.*

Lemma 7. *Let $\epsilon > 0$ be any fixed constant. Then for any large enough n , the value of the function $f : [\ln 3, \ln n] \rightarrow \mathbb{R}$, defined as $f(x) = e^x/x \cdot n^{\ln x/x}$, is bounded from below by $f(x) \geq n^{(1-\epsilon)\sqrt{2 \cdot \ln \ln n / \ln n}}$.*

Proof. We shall show the lemma by reasoning about the derivative of f . We will see that it contains two parts: one that is strictly convex, and the other that is strictly concave. This will allow us to conveniently reason about the sign of the derivative, and where the derivative vanishes. As a result, we will ensure that there is only one local minimum of f in the interior of the domain. An additional argument will ascertain that the values of f at the boundary are larger than the minimum value attained in the interior.

Let us investigate where the derivative

$$\frac{\partial f}{\partial x} = e^x n^{\ln x/x} / x^3 \cdot (x^2 - x + (1 - \ln x) \ln n)$$

vanishes. It happens only for such x , for which the parabola $x \mapsto x^2 - x$ “overlaps” the logarithmic plot $x \mapsto \ln n \ln x - \ln n$. We notice that the parabola is strictly convex, while the logarithmic plot is strictly concave. Therefore, we conclude that one of the three cases must happen: plots do not overlap, plots overlap at a single point, or plots overlap at exactly two distinct points. We shall see that the later must occur for any large enough n .

We will see that the plots overlap at exactly two points. Note that when $x = \ln 3$, then the value of the logarithmic plot is negative, while the value of the parabola is positive. Hence the parabola is “above” the logarithmic plot at the point $x = \ln 3$ of the domain. Similarly, it is “above” the logarithmic plot at the point $x = \ln n$, because for this x the highest order summand for the parabola is $\ln^2 n$, while it is only $\ln n \ln \ln n$ for the logarithmic plot. Finally, we observe that when $x = \sqrt{\ln n}$, then the plots are “swapped”: the logarithmic plot is “above” the parabola, because for this x the highest order summand for the parabola is $\ln n$, while the highest order summand for the logarithmic plot is as much as $1/2 \ln n \ln \ln n$. Therefore, for any large enough n , the plots must cross at exactly two points in the interior of the domain.

Now we are ready to evaluate the monotonicity of f . By inspecting the sign of the derivative, we conclude that f increases from $x = \ln 3$ until the first point, then it decreases until the second point, and then it increases again until $x = \ln n$. This holds for any large enough n .

This pattern of monotonicity allows us to bound from below the value of f in the interior of the domain. The function f attains a local minimum at the second point, and Lemma 6 teaches us that this point is in the range between $x_1 = \sqrt{1/2 \ln n \ln \ln n}$ and $x_2 = \sqrt{(1+\epsilon)/2 \ln n \ln \ln n}$. For large enough n , we can bound the value of the local minimum from below by $f_1 = e^{x_1}/x_2 \cdot n^{\ln x_1/x_2}$. We can further weaken this bound as

$$\begin{aligned} f_1 &= n^{-\ln x_2 / \ln n + \ln x_1 / x_2 + x_1 / \ln n} \geq n^{-\ln x_2 / \ln n + 1/2 \ln \ln n / x_2 + \sqrt{1/2 \ln \ln n / \ln n}} \\ &\geq n^{(1-\epsilon)\sqrt{2 \cdot \ln \ln n / \ln n}}, \end{aligned}$$

where the first inequality holds because for large enough n , $\ln(1/2 \ln \ln n)$ is positive, while the second inequality holds because for large enough n , $\ln x_2 \leq \ln \ln n$, and $1/\sqrt{1+\epsilon} \geq 1-\epsilon$, and for large enough n , $\sqrt{1/2 \ln \ln n / \ln n} - \ln \ln n / \ln n$ is larger than $\sqrt{1/(2+2\epsilon) \ln \ln n / \ln n}$.

Finally, we note that the values attained by f at the boundary are strictly larger than the value attained at the second point. Indeed, $f(\ln n)$ is strictly greater, because the function strictly increases from the second point towards $\ln n$. In addition, $f(\ln 3)$ is strictly greater because it is at least $n^{1.08}$, while the value attained at the second point is bounded from above by n raised to a power that tends to 0 as n tends to ∞ (in fact it suffices to see that the exponent of n in the bound on f_1 above, tends to 0 as n tends to ∞).

This completes the argument showing a lower bound on f .

The following two theorems show that we can construct an instance of AWT that has the exponent for n arbitrarily close to the exponent that is required, provided that we choose the value of q carefully enough.

Theorem 1. *Let $\epsilon > 0$ be any fixed constant. Then for any n that is large enough, any instance of the AWT algorithm for $p = n$ processors and n cells has work at least $n^{1+(1-\epsilon)\sqrt{2 \ln \ln n / \ln n}}$.*

Proof. This theorem is proven by combining the results shown in the preceding lemmas. Take any AWT algorithm for n cells and $p = n$ processors instantiated with a list R_q of q permutations on $[q]$. By Lemma 4, work of the instance is bounded from below by the expression $q^2 / (3 \text{Cont}(R_q)) \cdot n^{1+\log_q(\text{Cont}(R_q)/q)}$. By Lemma 5, we know that this expression does not increase when we replace $\text{Cont}(R_q)$ with a number that is smaller or equal to $\text{Cont}(R_q)$. Indeed, this is what we will do. By Lemma 1, we know that the value of $\text{Cont}(R_q)$ is bounded from below by $q \ln q$. Hence work of the AWT is at least $n/3 \cdot q / \ln q \cdot n^{\ln \ln q / \ln q}$.

Now we would like have a bound on this expression that does not depend on q . This bound should be fairly tight so that we can later find an instance of the AWT algorithm that has work close to the bound. Let us make a substitution $q = e^x$. We can use Lemma 7 with $\epsilon/2$ to bound the expression from below as desired, for large enough n , when q is in the range from 3 to n . What remains to be checked is how large work must be when the AWT algorithm is instantiated with just two permutations (i.e., when $q = 2$). In this case we know what contention of any list of two permutations is at least 3, and so work is bounded from below by n raised to a fixed power strictly greater than 1. Thus the lower bound holds for large enough n .

The following theorem explains that the lower bound can be nearly attained. The proof uses permutations described in Lemma 2. The proof is omitted.

Theorem 2. *Let $\epsilon > 0$ be any fixed constant. Then for any large enough m , when $q = \lceil e^{\sqrt{1/2 \ln m \ln \ln m}} \rceil$, and $h = \lceil \sqrt{2 \ln m / \ln \ln m} \rceil$, there exists an instance of the AWT algorithm for $p = n = q^h$ processors and n cells that has work at most $n^{1+(1+\epsilon)\sqrt{2 \ln \ln n / \ln n}}$.*

The above two theorems teach us that when q is selected carefully, we can create an instance of the AWT algorithm that is nearly optimal. A natural question that one immediately asks is: what if q is *not* selected well enough? Lemma 4 and Lemma 5 teach us that lower bound on work of an instance of the AWT algorithm depends on the number q of permutations on $[q]$ used by the instance. On one extreme, if q is a constant that is at least 2, then work must be at least n to some exponent that is greater than 1 and that is bounded away from 1. On the other extreme, if $q = n$, then work must be at least n^2 . In the “middle”, when q is about $e^{\sqrt{1/2 \ln n \ln \ln n}}$, then the lower bound is the weakest, and we can almost attain it as shown in the two theorems above. Suppose that we chose the value of q slightly away from the value $e^{\sqrt{1/2 \ln n \ln \ln n}}$. By how much must work be increased as compared to the lowest possible value of work? Although one can carry out a more precise analysis of the growth of a lower bound as a function of q , we will be contented with the following result, which already establishes a gap between the work possible to attain when q is chosen well, and the work required when q is not chosen well. The proof is omitted.

Proposition 1. *Let $r \geq 2$ be any fixed constant. For any large enough n , if the AWT algorithm is instantiated with q permutations on $[q]$, such that $16 \leq q \leq e^{\sqrt{1/2 \ln n \ln \ln n}/(r \cdot \ln \ln n)}$ or $e^{r \cdot \sqrt{1/2 \ln n \ln \ln n}} \leq q \leq n$, then its work is at least $n^{1+r/3} \cdot \sqrt{2 \ln n / \ln \ln n}$.*

4 Conclusions and Future Work

This paper shows how to create near-optimal instances of the Certified Write-All algorithm called AWT for n processors and n cells. We have seen that the choice of the number of permutation is critical for obtaining an instance of the AWT algorithm with near-optimal work. Specifically, when the algorithm is instantiated with about $e^{\sqrt{1/2 \ln n \ln \ln n}}$ permutations, then work of the instance can be near optimal, while when q is significantly away from $e^{\sqrt{1/2 \ln n \ln \ln n}}$, then work of any instance of the algorithm with this displaced q must be considerably higher than otherwise.

There are several follow-up research directions which will be interesting to explore. Any AWT algorithm has a progress tree with internal nodes of fanout q . One could consider generalized AWT algorithms where fanout does not need to be uniform. Suppose that a processor that visits a node of height i , uses a collection $R_{q(i)}^i$ of $q(i)$ permutations on $[q(i)]$. Now we could choose different values of $q(i)$ for different heights i . Does this technique enable any improvement of work as compared to the case when $q = q(1) = \dots = q(h)$? What are the best values for $q(1), \dots, q(h)$ as a function of n ? Suppose that we are given a relative cost κ of performing a write to the cell of the array w , compared to the cost of executing any other basic action. What is the shape of the progress tree that minimizes work? These questions give rise to more complex optimization problems, which would be interesting to solve.

The author developed a result related to the study presented in this paper. Specifically, the author showed a work-optimal deterministic algorithm for the asynchronous CWA problem for a nontrivial number of processors $p \ll n$. An extended abstract of this study will appear as [12], and a full version will appear in the upcoming doctoral dissertation of the author.

Acknowledgements. The author would like to thank Charles Leiserson for an invitation to join the Supercomputing Technologies Group, and Dariusz Kowalski, Larry Rudolph, and Alex Shvartsman for their comments that improved the quality of the presentation.

References

1. Anderson, R.J., Woll, H.: Wait-free Parallel Algorithms for the Union-Find Problem. Extended version of the STOC'91 paper of the authors, November 1 (1994)
2. Anderson, R.J., Woll, H.: Algorithms for the Certified Write-All Problem. *SIAM Journal on Computing*, Vol. 26(5) (1997) 1277–1283 (Preliminary version: STOC'91)
3. Aumann, Y., Kadem, Z.M., Palem, K.V., Rabin, M.O.: Highly Efficient Asynchronous Execution of Large-Grained Parallel Programs. 34th IEEE Symposium on Foundations of Computer Science FOCS'93, (1993) 271–280
4. Buss, J., Kanellakis, P.C., Ragde, P.L., Shvartsman, A.A.: Parallel Algorithms with Processor Failures and Delays. *Journal of Algorithms*, Vol. 20 (1996) 45–86 (Preliminary versions: PODC'91 and Manuscript'90)
5. Chlebus, B., Dobrev, S., Kowalski, D., Malewicz, G., Shvartsman, A., Vrto, I.: Towards Practical Deterministic Write-All Algorithms. 13th Symposium on Parallel Algorithms and Architectures SPAA'01, (2001) 271–280
6. Groote, J.F., Hesselink, W.H., Mauw, S., Vermeulen, R.: An algorithm for the asynchronous write-all problem based on process collision. *Distributed Computing*, Vol. 14(2) (2001) 75–81
7. Kanellakis, P.C., Shvartsman, A.A.: Efficient Parallel Algorithms Can Be Made Robust. *Distributed Computing*, Vol. 5(4) 1992 201–217 (Preliminary version: PODC'89)
8. Kanellakis, P.C., Shvartsman, A.A.: *Fault-Tolerant Parallel Computation*. Kluwer Academic Publishers (1997)
9. Kadem, Z.M., Palem, K.V., Raghunathan, A., Spirakis, P.G.: Combining Tentative and Definite Executions for Very Fast Dependable Parallel Computing. 23rd ACM Symposium on Theory of Computing STOC'91 (1991) 381–390
10. Knuth, D.E.: *The Art of Computer Programming Vol. 3* (third edition). Addison-Wesley Pub Co. (1998)
11. Lovász, L.: *Combinatorial Problems and exercises*, 2nd edition. North-Holland Pub. Co, (1993)
12. Malewicz, G.: A Work-Optimal Deterministic Algorithm for the Asynchronous Certified Write-All Problem. 22nd ACM Symposium on Principles of Distributed Computing PODC'03, (2003) to appear
13. Martel, C., Park, A., Subramonian, R.: Work-optimal asynchronous algorithms for shared memory parallel computers. *SIAM Journal on Computing*, Vol. 21(6) (1992) 1070–1099 (Preliminary version: FOCS'90)
14. Naor, J., Roth, R.M.: Constructions of Permutation Arrays for Certain Scheduling Cost Measures. *Random Structures and Algorithms*, Vol. 6(1) (1995) 39–50

I/O-Efficient Undirected Shortest Paths

Ulrich Meyer^{1,*} and Norbert Zeh^{2,**}

¹ Max-Planck-Institut für Informatik, Stuhlsatzhausenweg 85, 66123 Saarbrücken, Germany.

² Faculty of Computer Science, Dalhousie University, 6050 University Ave, Halifax, NS B3H 1W5, Canada.

Abstract. We present an I/O-efficient algorithm for the single-source shortest path problem on undirected graphs $G = (V, E)$. Our algorithm performs $\mathcal{O}(\sqrt{VE/B} \log_2(W/w) + \text{sort}(V + E) \log \log(VB/E))$ I/Os¹, where $w \in \mathbb{R}^+$ and $W \in \mathbb{R}^+$ are the minimal and maximal edge weights in G , respectively. For uniform random edge weights in $(0, 1]$, the expected I/O-complexity of our algorithm is $\mathcal{O}(\sqrt{VE/B} + ((V + E)/B) \log_2 B + \text{sort}(V + E))$.

1 Introduction

The *single-source shortest path (SSSP) problem* is a fundamental combinatorial optimization problem with numerous applications. It is defined as follows: Let G be a graph, let s be a distinguished vertex of G , and let ω be an assignment of non-negative real *weights* to the edges of G . The weight of a path is the sum of the weights of its edges. We want to find for every vertex v that is reachable from s , the weight $\text{dist}(s, v)$ of a minimum-weight (“shortest”) path from s to v .

The SSSP-problem is well-understood as long as the whole problem fits into *internal memory*. For larger data sets, however, classical SSSP-algorithms perform poorly, at least on sparse graphs: Due to the unpredictable order in which vertices are visited, the data is moved frequently between fast internal memory and slow *external memory*; the I/O-communication becomes the bottleneck.

I/O-model and previous results. We work in the standard I/O-model with one (logical) disk [1]. This model defines the following parameters:² N is the number of vertices and edges of the graph ($N = V + E$), M is the number of vertices/edges that fit into internal memory, and B is the number of vertices/edges that fit into a disk block. We assume that $2B < M < N$. In an *Input/Output* operation (or *I/O* for short), one block of data is transferred between disk and internal memory. The measure of performance of an algorithm is the number of I/Os it performs. The number of I/Os needed to read N contiguous items from disk is $\text{scan}(N) = \Theta(N/B)$. The number of I/Os required to

* Partially supported by EU programme IST-1999-14186 and DFG grant SA 933/1-1.

** Part of this work was done while visiting the Max-Planck-Institut in Saarbrücken.

¹ $\text{sort}(N) = \Theta((N/B) \log_{M/B}(N/B))$ is the I/O-complexity of sorting N data items.

² We use V and E to denote the vertex and edge sets of G as well as their sizes.

sort N items is $\text{sort}(N) = \Theta((N/B) \log_{M/B}(N/B))$ [1]. For all realistic values of N , B , and M , $\text{scan}(N) < \text{sort}(N) \ll N$.

External-memory graph algorithms have received considerable attention in recent years; see the surveys of [10,13]. Despite these efforts, only little progress has been made on the SSSP-problem: The best known lower bound is $\Omega(\text{sort}(V+E))$ I/Os, while the currently best algorithm, by Kumar and Schwabe [8], performs $\mathcal{O}(V + (E/B) \log_2(V/B))$ I/Os. For $E = \mathcal{O}(V)$, this is hardly better than naïvely running Dijkstra’s internal-memory algorithm [6,7] in external memory, which would take $\mathcal{O}(V \log_2 V + E)$ I/Os. Improved external-memory SSSP-algorithms exist for restricted graph classes such as planar graphs, grid graphs, and graphs of bounded treewidth; see [14] for an overview.

A number of improved internal-memory SSSP-algorithms have been proposed for bounded integer/float weights or bounded ratio W/w , where $w \in \mathbb{R}^+$ and $W \in \mathbb{R}^+$ are the minimal and maximal edge weights in G , respectively; see [11, 12] for an overview. For $W/w = 1$, SSSP becomes *breadth-first search* (BFS). A simple extension of the first $o(V)$ -I/O algorithm for undirected BFS [9] yields an SSSP-algorithm that performs $\mathcal{O}(\sqrt{VEW/B} + W \cdot \text{sort}(V+E))$ I/Os for integer weights in $[1, W]$. Obviously, W must be significantly smaller than B for this algorithm to be efficient. Furthermore, the algorithm requires that $BW < M$.

The only paper addressing the *average-case* I/O-complexity of SSSP [5] is restricted to random graphs with random edge weights. It reduces the I/O-complexity exclusively by exploiting the power of independent parallel disks; on a single disk, the performance of the algorithm is no better than that of [8].

New Results. We propose a new SSSP-algorithm for undirected graphs. The I/O-complexity of our algorithm is $\mathcal{O}(\sqrt{(VE/B) \log_2(W/w)} + \text{sort}(V+E))$ with high probability or $\mathcal{O}(\sqrt{(VE/B) \log_2(W/w)} + \text{sort}(V+E) \log \log(VB/E))$ deterministically, where $w \in \mathbb{R}^+$ and $W \in \mathbb{R}^+$ are the minimal and maximal edge weights in G , respectively. Compared to the solution of [9], the new algorithm exponentially increases the range of efficiently usable edge weights, while only requiring that $M = \Omega(B)$.³ These results hold for arbitrary graph structures and edge weights between w and W . For uniform random edge weights in $(0, 1]$, the average-case I/O-complexity of our algorithm reduces to $\mathcal{O}(\sqrt{VE/B} + ((V+E)/B) \log_2 B + \text{sort}(V+E))$. For sparse graphs, this matches the I/O-bound of the currently best BFS-algorithm.

2 Preliminaries and Outline

As previous I/O-efficient SSSP-algorithms [8,9], our algorithm is an I/O-efficient version of Dijkstra’s algorithm [6]. Dijkstra’s algorithm uses a priority queue Q to store all vertices of G that have not been settled yet (a vertex is said to be *settled* when its distance from s has been determined); the priority of a vertex v in Q is the length of the currently shortest known path from s to v . Vertices

³ In this extended abstract, we assume that $M = \Omega(B \log_2(W/w))$, to simplify the exposition.

are settled one-by-one by increasing distance from s . The next vertex v to be settled is retrieved from Q using a `DELETEMIN` operation. Then the algorithm relaxes the edges between v and all its non-settled neighbors, that is, performs a `DECREASEKEY`($w, \text{dist}(s, v) + \omega(v, w)$) operation for each such neighbor w whose priority is greater than $\text{dist}(s, v) + \omega(v, w)$.

An I/O-efficient version of Dijkstra’s algorithm has to (a) avoid accessing adjacency lists at random, (b) deal with the lack of optimal `DECREASEKEY` operations in current external-memory priority queues, and (c) efficiently remember settled vertices. The previous SSSP-algorithms of Kumar and Schwabe [8], **KS** for short, and Mehlhorn and Meyer [9], **MM**, address these issues as follows: **KS** ignores (a) and spends $\Omega(1)$ I/Os on retrieving the adjacency list of each settled vertex. **MM**, on the other hand, forms clusters of vertices and loads the adjacency lists of all vertices in a cluster into a “hot pool” of edges as soon as the first vertex in the cluster is settled. In order to relax the edges incident to settled vertices, the hot pool is scanned and all relevant edges are relaxed.

As for (b), **KS** uses a tournament tree, whereas **MM** applies a cyclic bucket queue composed of $2W + 1$ lists. Both support batched processing and emulate `INSERT` and `DECREASEKEY` operations using a weaker `UPDATE` operation, which decreases the priority of the element if it is already stored in the priority queue and otherwise inserts the element into the priority queue.

As for (c), **KS** performs an `UPDATE` operation for *every* neighbor of a settled vertex, which eliminates the need to remember previously settled vertices, but may re-insert settled vertices into the priority queue Q . Kumar and Schwabe call the latter a *spurious update*. Using a second priority queue Q^* , these re-inserted vertices are removed from Q before they can be settled for a second time.⁴ In contrast, **MM** deals with (c) by using half of its lists to identify settled vertices; `UPDATE` operations are performed only for non-settled vertices.

Our new approach inherits ideas from both algorithms: As **KS**, we use a second priority queue to eliminate the effect of spurious updates. But we replace the tournament tree used by **KS** with a hierarchical bucket queue (Section 3), which, in a way, is an I/O-efficient version of the integer priority queue of [2]. Next we observe that the relaxation of edges of large weight can be delayed because if such an edge is on a shortest path, it takes some time before its other endpoint is settled. Hence, we extend **MM**’s combination of clustering and hot pools to use a hierarchy of hot pools and gather long edges in hot pools that are touched much less frequently than the pools containing short edges. As we show in the full paper, already this idea alone works well on graphs with random edge weights. We obtain a worst-case guarantee for the I/O-complexity of our algorithm by storing even short edges in pools that are touched infrequently; we shift these edges to pools that are touched more and more frequently the closer the time of their relaxation draws. To make this work, we form clusters in a locality-preserving manner, essentially guaranteeing that a vertex is closer to its neighbors in the same cluster than to its neighbors in other clusters (Section 4.1).

⁴ The algorithm of [8] does not handle adjacent vertices with the same distance from s correctly. In the full paper, we provide a correct solution to this problem.

To predict the time of relaxation of every edge during the shortest path phase of the algorithm (Section 4.2), we use an explicit representation of the structure of each cluster, which is computed during the clustering phase.

Our clustering approach is similar to the one used in Thorup's linear-time SSSP-algorithm [12]. However, the precise definition of the clusters and their use during the shortest path phase of the algorithm differ from Thorup's, mainly because our goals are different. While we try to avoid random accesses by clustering nodes and treating their adjacency lists as one big list, Thorup's goal is to beat the sorting bound inherent in Dijkstra's algorithm by relaxing the order in which vertices are visited. Arguably, this makes the order in which the vertices are visited even more random.

3 An Efficient Batched Integer Priority Queue

In this section, we describe a simple batched integer priority queue Q , which can be seen as an I/O-efficient version of the integer priority queue of [2]. It supports UPDATE, DELETE, and BATCHEDDELETETEMIN operations. The first two operations behave as on a tournament tree; the latter retrieves *all* elements with minimal priority from Q . For the correctness of our data structure, the priority of an inserted or updated element has to be greater than the priority of the elements retrieved by the last BATCHEDDELETETEMIN operation. Let C be a bound so that, at all times, the difference between the minimum and maximum priorities of the elements in Q is at most C . Then Q supports the above operations in $\mathcal{O}((\log_2 C + \log_{M/B}(N/B))/B)$ I/Os amortized.

Q consists of $r = 1 + \log_2 C$ buckets. Each such bucket is represented by two sub-buckets $\mathcal{B}_i$ and $\mathcal{U}_i$. The buckets are defined by splitter elements $s_0 \leq s_1 \leq \dots \leq s_r = \infty$. Every entry (x, p_x) in $\mathcal{B}_i$, representing an element x with priority p_x , satisfies $s_{i-1} \leq p_x < s_i$. Initially, we set $s_0 = 0$ and, for $1 \leq i < r$, $s_i = 2^{i-1}$. We refer to $s_i - s_{i-1}$ as the *size* of bucket $\mathcal{B}_i$. These bucket sizes may change during the algorithm; but we enforce that, at all times, bucket $\mathcal{B}_1$ has size at most 1, and bucket $\mathcal{B}_i$, $1 < i < r$, has size 0 or a size between $2^{i-2}/3$ and 2^{i-2} . We use buckets $\mathcal{U}_1, \dots, \mathcal{U}_r$ to perform updates in a batched manner. In particular, bucket $\mathcal{U}_i$ stores updates to be performed on buckets $\mathcal{B}_i, \dots, \mathcal{B}_r$.

An UPDATE or DELETE operation inserts itself into $\mathcal{U}_1$, augmented with a time stamp. A BATCHEDDELETETEMIN operation reports the contents of $\mathcal{B}_1$, after filling it with elements from $\mathcal{B}_2, \dots, \mathcal{B}_r$ as follows: We iterate over buckets $\mathcal{B}_1, \dots, \mathcal{B}_i$, applying the updates in $\mathcal{U}_1, \dots, \mathcal{U}_i$ to $\mathcal{B}_1, \dots, \mathcal{B}_i$, until we find the first bucket $\mathcal{B}_i$ that is non-empty after these updates. We split the priority interval of $\mathcal{B}_i$ into intervals for $\mathcal{B}_1, \dots, \mathcal{B}_{i-1}$, assign an empty interval to $\mathcal{B}_i$, and distribute the elements of $\mathcal{B}_i$ over $\mathcal{B}_1, \dots, \mathcal{B}_{i-1}$, according to their priorities.

To incorporate the updates in $\mathcal{U}_1, \dots, \mathcal{U}_i$ into $\mathcal{B}_1, \dots, \mathcal{B}_i$, we sort the updates in $\mathcal{U}_1$ by their target elements and time stamps and repeat the following for $1 \leq j \leq i$: We scan $\mathcal{U}_j$ and $\mathcal{B}_j$, to update the contents of $\mathcal{B}_j$. If a deletion in $\mathcal{U}_j$ matches an existing element in $\mathcal{B}_j$, we remove this element from $\mathcal{B}_j$. If an UPDATE(x, p_x) operation in $\mathcal{U}_j$ matches an element (x, p'_x) in $\mathcal{B}_j$ and $p_x < p'_x$, we

replace (x, p'_x) with (x, p_x) in $\mathcal{B}_j$. If element x is not in $\mathcal{B}_j$, but $s_{j-1} \leq p_x < s_j$, we insert (x, p_x) into $\mathcal{B}_j$. If there are UPDATE and DELETE operations matching the same element in $\mathcal{B}_j$, we decide by the time stamps which action is to be taken. After these updates, we copy appropriate entries to $\mathcal{U}_{j+1}$, maintaining their sorted order: We scan $\mathcal{U}_j$ and $\mathcal{U}_{j+1}$ and insert every UPDATE(x, p_x) operation in $\mathcal{U}_j$ with $p_x \geq s_j$ into $\mathcal{U}_{j+1}$; for every DELETE(x) or UPDATE(x, p_x) operation with $p_x < s_j$, we insert a DELETE(x) operation into $\mathcal{U}_{j+1}$. (The latter ensures that UPDATE operations do not re-insert elements already in Q .)

To compute the new priority intervals for $\mathcal{B}_1, \dots, \mathcal{B}_{i-1}$, we scan $\mathcal{B}_i$ and find the smallest priority p of the elements in $\mathcal{B}_i$; we define $s_0 = p$ and, for $1 \leq j \leq i-1$, $s_j = \min\{p + 2^{j-1}, s_i\}$. Note that every $\mathcal{B}_j$, $1 \leq j < i$, of non-zero size has size 2^{j-2} , except the last such $\mathcal{B}_h$, whose size can be as small as 1. If the size of $\mathcal{B}_h$ is less than $2^{h-2}/3$, we redefine $s_{h-1} = s_h - 2^{h-2}/3$; this increases the size of $\mathcal{B}_h$ to $2^{h-2}/3$ and leaves $\mathcal{B}_{h-1}$ with a size between $2^{h-3}/3$ and 2^{h-3} .

To distribute the elements of $\mathcal{B}_i$ over $\mathcal{B}_1, \dots, \mathcal{B}_{i-1}$, we repeat the following for $j = i, i-1, \dots, 2$: We scan $\mathcal{B}_j$, remove all elements that are less than s_{j-1} from $\mathcal{B}_j$, and insert them into $\mathcal{B}_{j-1}$.

The I/O-complexity of an UPDATE or DELETE operation is $\mathcal{O}(1/B)$ amortized, because these operations only insert themselves into $\mathcal{U}_1$. To analyze the I/O-complexity of a BATCHEDDELETETMIN operation, we observe that every element is involved in the sorting of $\mathcal{U}_1$ only once; this costs $\mathcal{O}(\log_{M/B}(N/B)/B)$ I/Os amortized per element. When filling empty buckets $\mathcal{B}_1, \dots, \mathcal{B}_{i-1}$ with the elements of $\mathcal{B}_i$, every element in $\mathcal{B}_i$ moves down at least one bucket and will never move to a higher bucket again. If an element from $\mathcal{B}_i$ moves down x buckets, it is scanned $1+x$ times. Therefore, the total number of times an element from $\mathcal{B}_1, \dots, \mathcal{B}_r$ can be scanned before it reaches $\mathcal{B}_1$ is at most $2r = \mathcal{O}(\log_2 C)$. This costs $\mathcal{O}((\log_2 C)/B)$ I/Os amortized per element.

Emptying bucket $\mathcal{U}_i$ involves the scanning of buckets $\mathcal{U}_i, \mathcal{U}_{i+1}$, and $\mathcal{B}_i$. In the full paper, we prove that every element in $\mathcal{U}_i$ and $\mathcal{U}_{i+1}$ is involved in at most two such emptying processes of $\mathcal{U}_i$ before it moves to a higher bucket; every element in $\mathcal{B}_i$ is involved in only one such emptying process before it moves to a lower bucket. By combining our observations that every element is involved in the sorting of $\mathcal{U}_1$ at most once and that every element is touched only $\mathcal{O}(1)$ times per level in the bucket hierarchy, we obtain the following lemma.

Lemma 1. *There exists an integer priority queue Q that processes a sequence of N UPDATE, DELETE, and BATCHEDDELETETMIN operations in $\mathcal{O}(\text{sort}(N) + (N/B) \log_2 C)$ I/Os, where C is the maximal difference between the priorities of any two elements stored simultaneously in the priority queue.*

The following lemma, proved in the full paper, follows from the lower bound on the sizes of non-empty buckets. It is needed by our shortest path algorithm.

Lemma 2. *Let p be the priority of the entries retrieved by a BATCHEDDELETETMIN operation, and consider the sequence of all subsequent BATCHEDDELETETMIN operations that empty buckets $\mathcal{B}_h$, $h \geq i$, for some $i \geq 2$. Let $p_1, p_2, \dots$ be the priorities of the entries retrieved by these operations. Then $p_j - p \geq (j-4)2^{i-2}/3$.*

Note that we do not use that the priorities of the elements in Q are integers. Rather, we exploit that, if these priorities are integers, then $p > p'$ implies $p \geq p' + 1$, which in turn implies that after removing elements from $\mathcal{B}_1$, all subsequent insertions go into $\mathcal{B}_2, \dots, \mathcal{B}_r$. Hence, we can also use Q for elements with real priorities, as long as BATCHEDDELETETEMIN operations are allowed to produce a weaker output and UPDATE operations satisfy a more stringent constraint on their priorities. In particular, it has to be sufficient that the elements retrieved by a BATCHEDDELETETEMIN operation include all elements with smallest priority $p_{\min}$, their priorities are smaller than the priorities of all elements that remain in Q , and the priorities of any two retrieved elements differ by at most 1. Every subsequent UPDATE operation has to have priority at least $p_{\min} + 1$.

4 The Shortest Path Algorithm

Similar to the BFS-algorithm of [9], our algorithm consists of two phases: The *clustering phase* computes a partition of the vertex set of G into $o(V)$ vertex clusters $V_1, \dots, V_q$ and groups the adjacency lists of the vertices in these clusters into *cluster files* $\mathcal{F}_1, \dots, \mathcal{F}_q$. During the *shortest path phase*, when a vertex v is settled, we do not only retrieve its adjacency list but the whole cluster file from disk and store it in a collection of hot pools $\mathcal{H}_1, \dots, \mathcal{H}_r$. Thus, whenever another vertex in the same cluster as v is settled, it suffices to search the hot pools for its adjacency list. Using this approach, we perform only one random access per cluster instead of performing one random access per vertex to retrieve adjacency lists. The efficiency of our algorithm depends on how efficiently the edges incident to a settled vertex can be located in the hot pools and relaxed.

In Section 4.1, we show how to compute a *well-structured cluster partition*, whose properties help to make the shortest path phase, described in Section 4.2, efficient. In Section 4.3, we analyze the average-case complexity of our algorithm.

4.1 The Clustering Phase

In this section, we define a well-structured cluster partition $\mathcal{P} = (V_1, \dots, V_q)$ of G and show how to compute it I/O-efficiently. We assume w.l.o.g. that the minimum edge weight in G is $w = 1$. We group the edges of G into $r = \lceil \log_2 W \rceil$ *categories* so that the edges in category i have weight between 2^{i-1} and 2^i . The category of a vertex is the minimum of the categories of its incident edges. Let $G_0, \dots, G_r$ be a sequence of graphs defined as $G_0 = (V, \emptyset)$ and, for $1 \leq i \leq r$, $G_i = (V, E_i)$ with $E_i = \{e \in E : e \text{ is in category } j \leq i\}$. We call the connected components of G_i *category- i components*. The category of a cluster V_j is the smallest integer i so that V_j is completely contained in a category- i component. The *diameter* of V_j is the maximal distance in G between any two vertices in V_j . For some $\mu \geq 1$ to be fixed later, we call $\mathcal{P} = (V_1, \dots, V_q)$ *well-structured* if (P1) $q = \mathcal{O}(V/\mu)$, (P2) no vertex v in a category- i cluster V_j has an incident category- k edge (v, u) with $k < i$ and $u \notin V_j$, and (P3) no category- i cluster has diameter greater than $2^i \mu$.

The goal of the clustering phase is to compute a well-structured cluster partition $\mathcal{P} = (V_1, \dots, V_q)$ along with *cluster trees* $\tilde{T}_1, \dots, \tilde{T}_q$ and *cluster files* $\mathcal{F}_1, \dots, \mathcal{F}_q$; the cluster trees capture the containment of the vertices in clusters $V_1, \dots, V_q$ in the connected components of graphs $G_0, \dots, G_r$; the cluster files are the concatenations of the adjacency lists of the vertices in the clusters.

Computing the cluster partition. We use a minimum spanning tree T of G to construct a well-structured cluster partition of G . For $0 \leq i \leq r$, let T_i be the subgraph of T that contains all vertices of T and all tree edges in categories $1, \dots, i$. Then two vertices are in the same connected component of T_i if and only if they are in the same connected component of G_i . Hence, a well-structured cluster partition of T is also a well-structured cluster partition of G . We show how to compute the former. For any set $X \subseteq V$, we define its *tree diameter* as the total weight of the edges in the smallest subtree of T that contains all vertices in X . We guarantee in fact that every category- i cluster in the computed partition has tree diameter at most $2^i \mu$. Since the tree diameter of a cluster may be much larger than its diameter, we may generate more clusters than necessary; but their number is still $\mathcal{O}(V/\mu)$.

We iterate over graphs $T_0, \dots, T_r$. In the i -th iteration, we partition the connected components of T_i into clusters. To bound the number of clusters we generate, we partition a component of T_i only if its tree diameter is at least $2^i \mu$ and it contains vertices that have not been added to any cluster in the first $i - 1$ iterations. We call these vertices *active*; a component is active if it contains at least one active vertex; an active category- i component is *heavy* if its tree diameter is at least $2^i \mu$. To partition a heavy component C of T_i into clusters, we traverse an Euler tour of C , forming clusters as we go. When we visit an active category- $(i - 1)$ component in C for the first time, we test whether adding this component to the current cluster would increase its tree diameter beyond $2^i \mu$. If so, we start a new cluster consisting of the active vertices in this component; otherwise, we add all active vertices in the component to the current cluster.

This computation takes $\mathcal{O}(\text{sort}(V + E) + (V/B) \log_2(W/w))$ I/Os w.h.p.: A minimum spanning tree T of G can be computed in $\mathcal{O}(\text{sort}(V + E))$ I/Os w.h.p. [4]. An Euler tour L of T can be computed in $\mathcal{O}(\text{sort}(V))$ I/Os [4]. The heavy components of a graph T_i can be identified and partitioned using two scans of L and using three stacks to keep track of the necessary information as we advance along L . Hence, one iteration of the clustering algorithm takes $\mathcal{O}(V/B)$ I/Os; all $r = \log_2 W$ iterations take $\mathcal{O}((V/B) \log_2 W)$ I/Os.

It remains to be argued that the computed partition is well-structured. Clearly, every category- i cluster has diameter at most $2^i \mu$. Such a cluster is completely contained in a category- i component, and no category- $(i - 1)$ component has vertices in two different category- i clusters. Hence, all clusters have Property (P2). In the full paper, we show that their number is $\mathcal{O}(V/\mu)$.

Lemma 3. *A well-structured cluster partition of a weighted graph $G = (V, E)$ can be computed in $\mathcal{O}(\text{sort}(V + E) + (V/B) \log_2(W/w))$ I/Os w.h.p.*

Computing the cluster trees. In order to decide in which hot pool to store an edge (v, w) during the shortest path phase, we must be able to find

the smallest i so that the category- i component containing v includes a settled vertex. Next we define *cluster trees* as the tool to determine category i efficiently.

The nesting of the components of graphs $T_0, \dots, T_r$ can be captured in a tree $\tilde{T}$. The nodes of $\tilde{T}$ represent the connected components of $T_0, \dots, T_r$. A node representing a category- i component C is the child of a node representing a category- $(i+1)$ component C' if $C \subseteq C'$. We ensure that every internal node of $\tilde{T}$ has at least two children; that is, a subgraph C of T that is a component of more than one graph T_i is represented only once in $\tilde{T}$. We define the category of such a component as the largest integer i so that C is a component of T_i .

Now we define the cluster tree $\tilde{T}_j$ for a category- i cluster V_j : Let C be the category- i component containing V_j , and let v be the node in $\tilde{T}$ that represents C ; $\tilde{T}_j$ consists of the paths in $\tilde{T}$ from v to all the leaves that represent vertices in V_j .

Tree $\tilde{T}$ can be computed in r scans of Euler tour L , similar to the construction of the clusters; this takes $\mathcal{O}((V/B) \log_2 W)$ I/Os. Trees $\tilde{T}_1, \dots, \tilde{T}_q$ can be computed in $\mathcal{O}(\text{sort}(V))$ I/Os, using a DFS-traversal of $\tilde{T}$. In the full paper, we show that their total size is $\mathcal{O}(V)$.

Computing the cluster files. The last missing piece of information about clusters $V_1, \dots, V_q$ is their *cluster files* $\mathcal{F}_1, \dots, \mathcal{F}_q$. File $\mathcal{F}_j$ is the concatenation of the adjacency lists of the vertices in V_j . Clearly, files $\mathcal{F}_1, \dots, \mathcal{F}_q$ can be computed in $\mathcal{O}(\text{sort}(V + E))$ I/Os, by sorting the edge set of G appropriately.

4.2 The Shortest Path Phase

At a very high level, the shortest path phase is similar to Dijkstra's algorithm. We use the integer priority queue from Section 3 to store all non-settled vertices; their priorities equal their tentative distances from s . We proceed in iterations: Each iteration starts with a BATCHEDDELETETEMIN operation, which retrieves the vertices to be settled in this iteration. The priorities of the retrieved vertices are recorded as their final distances from s , which is correct because all edges in G have weight at least 1 and the priorities of the retrieved vertices differ by at most 1. Finally, we relax the edges incident to the retrieved vertices.

We use the clusters built in the previous phase to avoid spending one I/O per vertex on retrieving adjacency lists. When the first vertex in a cluster V_j is settled, we load the whole cluster file $\mathcal{F}_j$ into a set of hot pools $\mathcal{H}_1, \dots, \mathcal{H}_r$. When we subsequently settle a vertex $v \in V_k$, we scan the hot pools to see whether they contain v 's adjacency list. If so, we relax the edges incident to v ; otherwise, we have to load file $\mathcal{F}_k$ first. Since we load every cluster file only once, we spend only $\mathcal{O}(V/\mu + E/B)$ I/Os on retrieving adjacency lists. Since we scan the hot pools in each iteration, to decide which cluster files need to be loaded, the challenge is to avoid touching an edge too often during these scans. We solve this problem by using a hierarchy of hot pools $\mathcal{H}_1, \dots, \mathcal{H}_r$ and inspecting only a subset $\mathcal{H}_1, \dots, \mathcal{H}_i$ of these pools in each iteration. We choose the pool where to store every edge to be the highest pool that is scanned at least once before this edge has to be relaxed. The precise choice of this pool is based on the following two observations: (1) It suffices to relax a category- i

edge incident to a settled vertex v any time before the first vertex at distance at least $\text{dist}(s, v) + 2^{i-1}$ from s is settled. (2) An edge in a category- i component cannot be up for relaxation before the first vertex in the component is about to be settled. The first observation allows us to store all category- i edges in pool $\mathcal{H}_i$, as long as we guarantee that $\mathcal{H}_i$ is scanned at least once between the settling of two vertices whose distances from s differ by at least 2^{i-1} . The second observation allows us to store even category- j edges, $j < i$, in pool $\mathcal{H}_i$, as long as we move these edges to lower pools as the time of their relaxation approaches.

The second observation is harder to exploit than the first one because it requires some mechanism to identify for every vertex v , the smallest category i so that the category- i component containing v contains a settled vertex or a vertex to be settled soon. We provide such a mechanism using four additional pools $\mathcal{V}_i$, $\mathcal{T}_i$, $\mathcal{H}'_i$, and $\mathcal{T}'_i$ per category. Pool $\mathcal{V}_i$ contains settled vertices whose category- j edges, $j < i$, have been relaxed. Pools $\mathcal{T}_1, \dots, \mathcal{T}_r$ store the nodes of the cluster trees corresponding to the cluster files loaded into pools $\mathcal{H}_1, \dots, \mathcal{H}_r$. A cluster tree node is stored in pool $\mathcal{T}_i$ if its corresponding component C is in category i or it is in category $j < i$ and the smallest component containing C and at least one settled vertex or vertex in $\mathcal{B}_1, \dots, \mathcal{B}_i$ is in category i . We store an edge (v, w) in pool $\mathcal{H}_i$ if its category is i or it is less than i and the cluster tree node corresponding to vertex v resides in pool $\mathcal{T}_i$. Pools $\mathcal{H}'_1, \dots, \mathcal{H}'_r$ and $\mathcal{T}'_1, \dots, \mathcal{T}'_r$ are auxiliary pools that are used as temporary storage after loading new cluster files and trees and before we determine the correct pools where to store their contents. We maintain a labeling of the cluster tree nodes in pools $\mathcal{T}'_1, \dots, \mathcal{T}'_r$ that helps us to identify the pool $\mathcal{T}_i$ where each node in these pools is to be stored: A node is either marked or unmarked; a marked node in $\mathcal{T}'_i$ corresponds to a component that contains a settled vertex or a vertex in $\mathcal{B}_1, \dots, \mathcal{B}_i$. In addition, every node stores the category of (the component corresponding to) its lowest marked ancestor in the cluster tree.

To determine the proper subset of pools to be inspected in each iteration, we tie the updates of the hot pools and the relaxation of edges to the updates of priority queue buckets performed by the BATCHEDDELETETMIN operation. Every such operation can be divided into two phases: The *up-phase* incorporates the updates in buckets $\mathcal{U}_1, \dots, \mathcal{U}_i$ into buckets $\mathcal{B}_1, \dots, \mathcal{B}_i$; the *down-phase* distributes the contents of bucket $\mathcal{B}_i$ over buckets $\mathcal{B}_1, \dots, \mathcal{B}_{i-1}$. We augment the up-phase so that it loads cluster files and relaxes the edges in pools $\mathcal{H}_1, \dots, \mathcal{H}_i$ that are incident to settled vertices. In the down-phase, we shift edges from pool $\mathcal{H}_i$ to pools $\mathcal{H}_1, \dots, \mathcal{H}_{i-1}$ as necessary. The details of these two phases are as follows:

The up-phase. We update the contents of the category- j pools and relax the edges in $\mathcal{H}_j$ after applying the updates from $\mathcal{U}_j$ to $\mathcal{B}_j$. We mark every node in $\mathcal{T}'_j$ whose corresponding component contains a vertex in $\mathcal{B}_j \cup \mathcal{V}_j$ and identify, for every node, the category of its lowest marked ancestor in $\mathcal{T}'_j$. We move every node whose lowest marked ancestor is in a category greater than j to $\mathcal{T}'_{j+1}$ and insert the other nodes into $\mathcal{T}_j$. For every leaf of a cluster tree that was moved to $\mathcal{T}'_{j+1}$, we move the whole adjacency list of the corresponding vertex from $\mathcal{H}'_j$ to $\mathcal{H}'_{j+1}$. Any other edge in $\mathcal{H}'_j$ is moved to $\mathcal{H}'_{j+1}$ if its category is greater than j ;

otherwise, we insert it into $\mathcal{H}_j$. We scan $\mathcal{V}_j$ and $\mathcal{H}_j$ to identify all category- j vertices in $\mathcal{V}_j$ that do not have incident edges in $\mathcal{H}_j$, load the corresponding cluster files and trees into $\mathcal{H}'_j$ and $\mathcal{T}'_j$, respectively, and sort $\mathcal{H}'_j$ and $\mathcal{T}'_j$. We proceed as above to decide where to move the nodes and edges in $\mathcal{T}'_j$ and $\mathcal{H}'_j$. We scan $\mathcal{V}_j$ and $\mathcal{H}_j$ again, this time to relax all edges in $\mathcal{H}_j$ incident to vertices in $\mathcal{V}_j$. As we argue below, the resulting UPDATE operations affect only $\mathcal{B}_{j+1}, \dots, \mathcal{B}_r$; so we insert these updates into $\mathcal{U}_{j+1}$. Finally, we move all vertices in $\mathcal{V}_j$ to $\mathcal{V}_{j+1}$ and either proceed to $\mathcal{B}_{j+1}$ or enter the down-phase with $i = j$, depending on whether or not $\mathcal{B}_j$ is empty.

The down-phase. We move edges and cluster tree nodes from $\mathcal{H}_j$ and $\mathcal{T}_j$ to $\mathcal{H}_{j-1}$ and $\mathcal{T}_{j-1}$ while moving vertices from bucket $\mathcal{B}_j$ to bucket $\mathcal{B}_{j-1}$. First we identify all nodes in $\mathcal{T}_j$ whose corresponding components contain vertices that are pushed to $\mathcal{B}_{j-1}$. If the category of such a node v is less than j , we push the whole subtree rooted at v to $\mathcal{T}_{j-1}$. For every leaf that is pushed to $\mathcal{T}_{j-1}$, we push all its incident edges of category less than j from $\mathcal{H}_j$ to $\mathcal{H}_{j-1}$. Finally, we remove all nodes of $\tilde{T}$ from $\mathcal{T}_j$ that have no descendent leaves left in $\mathcal{T}_j$.

Correctness. We need to prove the following: (1) The relaxation of a category- i edge can only affect buckets $\mathcal{B}_{i+1}, \dots, \mathcal{B}_r$. (2) Every category- i edge (v, w) is relaxed before a vertex at distance at least $\text{dist}(s, v) + 2^{i-1}$ from s is settled.

To see that the first claim is true, observe that a vertex v that is settled between the last and the current relaxation of edges in $\mathcal{H}_i$ has distance at least $l - 2^{i-2}$ from s , where $[l, u)$ is the priority interval of bucket $\mathcal{B}_i$, i.e., $u \leq l + 2^{i-2}$. Since an edge $(v, w) \in \mathcal{H}_i$ has weight at least 2^{i-1} , we have $\text{dist}(s, v) + \omega(v, w) > l + 2^{i-2} = u$; hence, vertex w will be inserted into one of buckets $\mathcal{B}_{i+1}, \dots, \mathcal{B}_r$.

The second claim follows immediately if we can show that when vertex v reaches pool $\mathcal{V}_i$, edge (v, w) either is in $\mathcal{H}_i$ or is loaded into $\mathcal{H}_i$. This is sufficient because we have to empty at least one bucket $\mathcal{B}_j$, $j \geq i$, between the settling of vertex v and the settling of a vertex at distance at least $\text{dist}(s, v) + 2^{i-1}$. Since edge (v, w) is in category i , the category of vertex v is $h \leq i$. When $v \in \mathcal{V}_h$, the cluster file containing v 's adjacency list is loaded into pool $\mathcal{H}'_h$, and all category- h edges incident to v are moved to $\mathcal{H}_h$, unless pool $\mathcal{H}_h$ already contains a category- h edge incident to v . It is easy to verify that in the latter case, $\mathcal{H}_h$ must contain *all* category- h edges incident to v . This proves the claim for $i = h$. For $i > h$, we observe that the adjacency list of v is loaded at the latest when $v \in \mathcal{V}_h$. If this is the case, edge (v, w) is moved to pool $\mathcal{H}_i$ at the same time when vertex v reaches pool $\mathcal{V}_i$. If vertex v finds an incident category- h edge in $\mathcal{H}_h$, then edge (v, w) is either in one of pools $\mathcal{H}'_{h+1}, \dots, \mathcal{H}'_i$ or in one of pools $\mathcal{H}_i, \dots, \mathcal{H}_r$. In the former case, edge (v, w) is placed into pool $\mathcal{H}_i$ when vertex v reaches pool $\mathcal{V}_i$. In the latter case, edge (v, w) is in fact in pool $\mathcal{H}_i$ because, otherwise, pool $\mathcal{H}_h$ could not contain any edge incident to v . This proves the claim for $i > h$.

I/O-complexity. The analysis is based on the following two claims proved below: (1) Every cluster file is loaded exactly once. (2) Every edge is involved in $\mathcal{O}(\mu)$ updates of a pool $\mathcal{H}_i$ before it moves to a pool of lower category; the same is true for the cluster tree nodes in $\mathcal{T}_i$. In the full paper, we show that all the updates of the hot pools can be performed using a constant number of scans. Also

note that every vertex is involved in the sorting of pool $\mathcal{V}_1$ only once, and every edge or cluster tree node is involved in the sorting of a pool $\mathcal{H}'_i$ or $\mathcal{T}'_j$ only once. These observations together establish that our algorithm spends $\mathcal{O}(V/\mu + (V + E)/B)$ I/Os on loading cluster files and cluster trees; $\mathcal{O}(\text{sort}(V + E))$ I/Os on sorting pools $\mathcal{V}_1, \mathcal{H}'_1, \dots, \mathcal{H}'_r$, and $\mathcal{T}'_1, \dots, \mathcal{T}'_r$; and $\mathcal{O}((\mu E/B) \log_2 W)$ I/Os on all remaining updates of priority queue buckets and hot pools. Hence, the total I/O-complexity of the shortest path phase is $\mathcal{O}(V/\mu + (\mu E/B) \log_2 W + \text{sort}(V + E))$.

To show that every cluster file is loaded exactly once, we have to prove that once a cluster file containing the adjacency list of a category- i vertex v has been loaded, vertex v finds an incident category- i edge (v, w) in $\mathcal{H}_i$. The only circumstance possibly preventing this is if $(v, w) \in \mathcal{H}_j$, $j > i$, at the time when $v \in \mathcal{V}_i$. However, at the time when edge (v, w) was moved to $\mathcal{H}_j$, no vertex in the category- $(j - 1)$ component C that contains v had been settled or was in one of $\mathcal{B}_1, \dots, \mathcal{B}_{j-1}$. Every vertex in C that is subsequently inserted into the priority queue is inserted into a bucket $\mathcal{B}_{h+1}$, $h \geq j$, because this happens as the result of the relaxation of a category- h edge. Hence, before any vertex in C can be settled, such a vertex has to be moved from $\mathcal{B}_j$ to $\mathcal{B}_{j-1}$, which causes edge (v, w) to move to $\mathcal{H}_{j-1}$. This proves that vertex v finds edge (v, w) in $\mathcal{H}_i$.

To prove that every edge is involved in at most $\mathcal{O}(\mu)$ scans of pool $\mathcal{H}_i$, observe that at the time when an edge (v, w) is moved to pool $\mathcal{H}_i$, there has to be a vertex x in the same category- i component C as v that either has been settled already or is contained in one of buckets $\mathcal{B}_1, \dots, \mathcal{B}_i$ and hence will be settled before pool $\mathcal{H}_i$ is scanned for the next time; moreover, there has to be such a vertex whose distance from v is at most $2^i \mu$. By Lemma 2, the algorithm makes progress at least $2^{j-2}/3$ every time pool $\mathcal{H}_i$ is scanned. Hence, after $\mathcal{O}(\mu)$ scans, vertex v is settled, so that edge (v, w) is relaxed before or during the next scan of pool $\mathcal{H}_i$. This proves the second claim.

Summing the I/O-complexities of the two phases, we obtain that the I/O-complexity of our algorithm is $\mathcal{O}(V/\mu + (\mu(V + E)/B) \log_2 W + \text{sort}(V + E))$ w.h.p. By choosing $\mu = \sqrt{VB/(E \log_2 W)}$, we obtain the following result.

Theorem 1. *The single source shortest path problem on an undirected graph $G = (V, E)$ can be solved in $\mathcal{O}(\sqrt{(VE/B) \log_2(W/w)} + \text{sort}(V + E))$ I/Os w.h.p., where w and W are the minimal and maximal edge weights in G .*

Observe that the only place in the algorithm where randomization is used is in the computation of the minimum spanning tree. In [3], it is shown that a minimum spanning tree can be computed in $\mathcal{O}(\text{sort}(V + E) \log \log(VB/E))$ I/Os deterministically. Hence, we can obtain a deterministic version of our algorithm that takes $\mathcal{O}(\sqrt{(VE/B) \log_2(W/w)} + \text{sort}(V + E) \log \log(VB/E))$ I/Os.

4.3 An Average-Case Analysis

Next we analyze the average-case complexity of our algorithm. We assume uniform random edge weights in $(0, 1]$, but make no randomness assumption about the structure of the graph. In the full paper, we show that we can deal with “short

edges”, that is, edges whose weight is at most $1/B$, in expected $\mathcal{O}(\text{sort}(E))$ I/Os, because their expected number is E/B . We deal with long edges using the algorithm described in this section. Now we observe that the expected number of category- i edges in G and category- i nodes in $\tilde{T}$ is $\mathcal{O}(2^{i-1}E/B)$. Each such edge or node moves up through the hierarchy of pools $\mathcal{H}'_1, \dots, \mathcal{H}'_r$ or $\mathcal{T}'_1, \dots, \mathcal{T}'_r$, being touched $\mathcal{O}(1)$ times per category. Then it moves down through pools $\mathcal{H}_r, \mathcal{H}_{r-1}, \dots, \mathcal{H}_i$ or $\mathcal{T}_r, \mathcal{T}_{r-1}, \dots, \mathcal{T}_i$, being touched $\mathcal{O}(\mu)$ times per category. Hence, the total cost of scanning pools $\mathcal{H}'_1, \dots, \mathcal{H}'_r$ and $\mathcal{T}'_1, \dots, \mathcal{T}'_r$ is $\mathcal{O}(E \log_2 B/B)$, and the expected total cost of scanning pools $\mathcal{H}_1, \dots, \mathcal{H}_r$ and $\mathcal{T}_1, \dots, \mathcal{T}_r$ is $\mathcal{O}((\mu E/B^2) \sum_{i=1}^r 2^{i-1}(r-i+1)) = \mathcal{O}(\mu E/B)$. Thus, the expected I/O-complexity of our algorithm is $\mathcal{O}(V/\mu + \mu E/B + ((V+E)/B) \log_2 B + \text{sort}(V+E))$. By choosing $\mu = \sqrt{VB/E}$, we obtain the following result.

Theorem 2. *The single source shortest path problem on an undirected graph $G = (V, E)$ whose edge weights are drawn uniformly at random from $(0, 1]$ can be solved in expected $\mathcal{O}(\sqrt{VE/B} + ((V+E)/B) \log_2 B + \text{sort}(V+E))$ I/Os.*

References

1. A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Comm. of the ACM*, pp. 1116–1127, 1988.
2. R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37(2):213–233, 1990.
3. L. Arge, G. S. Brodal, and L. Toma. On external memory MST, SSSP, and multi-way planar separators. *Proc. 7th SWAT*, LNCS 1851, pp. 433–447. Springer, 2000.
4. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. *Proc. 6th ACM-SIAM SODA*, pp. 139–149, 1995.
5. A. Crauser, K. Mehlhorn, U. Meyer, and P. Sanders. A parallelization of Dijkstra’s shortest path algorithm. *Proc. 23rd MFCS*, LNCS 1450, pp. 722–731. Springer, 1998.
6. E. W. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269–271, 1959.
7. M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
8. V. Kumar and E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. *Proc. 8th IEEE SPDP*, pp. 169–176, 1996.
9. K. Mehlhorn and U. Meyer. External-memory breadth-first search with sublinear I/O. *Proc. 10th ESA*, LNCS 2461, pp. 723–73. Springer, 2002.
10. U. Meyer, P. Sanders, and J. F. Sibeyn, editors. *Algorithms for Memory Hierarchies*, LNCS 2625. Springer, 2003.
11. R. Raman. Recent results on the single-source shortest paths problem. *ACM SIGACT News*, 28(2):81–87, June 1997.
12. M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46:362–394, 1999.
13. J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.
14. N. Zeh. *I/O-Efficient Algorithms for Shortest Path Related Problems*. PhD thesis, School of Computer Science, Carleton University, 2002.

On the Complexity of Approximating TSP with Neighborhoods and Related Problems^{*}

Shmuel Safra and Oded Schwartz

Tel Aviv University, Tel Aviv 69978, Israel {safra,odedsc}@post.tau.ac.il

Abstract. We prove that various geometric covering problems, related to the Travelling Salesman Problem cannot be efficiently approximated to within any constant factor unless $P = NP$. This includes the Group-Travelling Salesman Problem (TSP with Neighborhoods) in the Euclidean plane, the Group-Steiner-Tree in the Euclidean plane and the Minimum Watchman Tour and the Minimum Watchman Path in 3-D. Some inapproximability factors are also shown for special cases of the above problems, where the size of the sets is bounded. Group-TSP and Group-Steiner-Tree where each neighbourhood is connected are also considered. It is shown that approximating these variants to within any constant factor smaller than 2, is NP-hard.

1 Introduction

The Travelling Salesman Problem (*TSP*) is a classical problem in combinatorial optimization, and has been studied extensively in many forms. It is the problem of a travelling salesman who has to visit n locations, returning eventually to the starting point. The goal may be to minimize the total distance traversed, driving time, or money spent on toll roads, where the cost (in terms of length units, time units money or other) is given by an $n \times n$ matrix of non-negative weights. In the geometric TSP, the matrix represents distances in a Euclidean space. In other certain natural instances (e.g, time and money) while weights might not agree with a Euclidean metric, they still obey the triangle inequality, namely the cost of traversing from a to b is not higher than the cost of traversing from a to b , via other points. Formally, the Travelling Salesman Problem can be defined as follows: given a set P of points in a metric space, find a traversal of shortest length visiting each point of P , and return to the starting point.

TSP in the Plane. Finding the optimal solution of a given instance of TSP with triangle inequality is NP-hard, as obtained by a simple reduction from the Hamilton-Cycle problem. Even in the special case where the matrix represents distances between points in the Euclidean plane, it is also proved to be NP-hard [GGJ76,Pap77]. The latter problem has a polynomial time approximation scheme (*PTAS*) - that is, although it is NP-hard to come up with the optimal solution, one can have, in polynomial time, a $1 + \varepsilon$ approximation of the optimal

^{*} Research supported in part by the Fund for Basic Research Administered by the Israel Academy of Sciences, and a Bikura grant.

solution for any $\varepsilon > 0$, (see [Aro96,Mit96]). This however, is not the case for the non-geometric variants.

Triangle Inequality. In the general case, approximating TSP to within any constant factor is NP-hard (again, by a simple reduction from the Hamilton-Cycle problem). When only triangle inequality is assured, the best known algorithm gives a $\frac{3}{2}$ approximation ratio, if weights are symmetric [Chr76]. If weights can be asymmetric (that is, the cost from a to b is not necessarily the same as the cost from b to a), the best known approximation ratio is $O(\log n)$ [FGM82]. Although the asymmetric case may seem unnatural having the Euclidean metric intuition in mind, when weights represent measures other than length, or for example - where the lengths are of one-way roads, the asymmetric formulation is natural. Both the symmetric and the asymmetric variances are conjectured to be efficiently approximable to within $\frac{4}{3}$ (see [CVM00]).

In regard to the hardness of approximation, Papadimitriou and Vempala [PV00] gave evidence that unless $P = NP$, the symmetric case cannot be efficiently approximated to within a factor smaller than $\frac{234}{233}$, and the asymmetric case to within a factor smaller than $\frac{98}{97}$. For bounded metrics Engebretsen and Karpinski [EK01] showed hardness of approximation factors of $\frac{131}{130}$ and $\frac{174}{173}$ respectively.

Group-TSP. A natural generalization of this problem is the Group-TSP (*G-TSP*), known also by the names the One-of-a-Set-TSP, TSP with neighborhoods and the Errand Scheduling problem. A travelling salesman has to meet n customers. Each of them is willing to meet the salesman in specified locations (referred to as a region). For instances in which each region contains exactly one point, this becomes the TSP problem. For instances in which all edges are of weight 1, this becomes the Hitting-Sets (or Set-Cover) problem. Another natural illustration of the *G-TSP* is the Errand Scheduling Problem as described in [Sla97]. A list of n jobs (or errands) to be carried out is given, each of which can be performed in a few locations. The objective is to find a close tour of minimal length, such that all jobs can be performed. That is, for every job on the list, there is at least one location on the tour, at which the job can be performed (it is allowed to perform more than one job in a single location). If every job can be performed in at most k locations, then we call this problem k -*G-TSP*. k -*G-TSP* (with symmetric weights) can be approximated to within $\frac{3k}{2}$ [Sla97]. This algorithm generalizes the $\frac{3}{2}$ approximation ratio of Christofides [Chr76] for $k \geq 1$.

As *G-TSP* (with triangle inequality) is a generalization of both TSP and Set-Cover inapproximability factor for any of those two problems, holds for the *G-TSP*. Thus, by [LY94,RS97,Fei98], *G-TSP* is hard to approximate to within a logarithmic factor. However this is not trivially true for the geometric variant of *G-TSP*.

G-TSP in the Plane. This problem was first studied by Arkin and Hassin [AH94] who gave a constant approximation ratio algorithm for it where the regions (or neighborhoods) are well behaved in some form (e.g. consist of disks, parallel segments of equal length and translates of a convex region). Mata and

Mitchell [MM95] and Gudmundsson and Levcopoulos [GL99] showed an $O(\log n)$ approximation ratio for arbitrary (possibly overlapping) polygonal regions. A constant factor approximation algorithm for the case where neighborhoods are disjoint convex fat objects was suggested by de Berg, Gudmundsson, Katz, Levcopoulos, Overmars, and van der Stappen [dBGK⁺02]. Recently Dumitrescu and Mitchell [DM01] gave a constant factor approximation algorithm for the case of arbitrary connected neighborhoods (i.e., path-wise connected) having comparable diameter, and a PTAS for the special case of pairwise disjoint unit disk neighborhoods. The best known approximation hardness result for this problem is of $\frac{391}{390} - \varepsilon \approx 1.003$ [dBGK⁺02].

Steiner Tree. Another related problem is the minimum Steiner spanning tree problem, or Steiner Tree problem (*ST*). A Steiner tree of S is a tree whose nodes contain the given set S . The nodes of the tree that are not the points of S are called Steiner points. Although finding a minimum spanning tree can be computed in polynomial time, the former problem is NP-hard. In the Euclidean case the problem remains NP-hard [GGJ77], but admits a PTAS [Aro96,Mit96].

Group Steiner Tree. The Steiner tree notion can be generalized similarly to the generalization of TSP to G-TSP. In the Group Steiner Tree Problem (*G-ST*) (also known as Class Steiner Problem, the Tree Cover Problem and the One-of-a-Set Steiner Problem) we are given an undirected graph with edge weights and subsets of the vertices. The objective is to find a minimum weighted tree, having at least one vertex of each subset.

As G-ST is another generalization of set cover (even when the weight function obeys triangle inequality) any approximation hardness factor for set-cover applies to G-ST [Ihl92]. Thus, by [LY94,RS97,Fei98], G-ST is hard to approximate within a logarithmic factor. As in G-TSP, this is not trivially true for the geometric domain. In 1997 Slavik [Sla97] gave an $O(\log n)$ approximation algorithm for a restricted case of this problem and a $2k$ approximation algorithm for the variant in which sets are of size at most k . For sets of unbounded size, no constant approximation algorithm is known, even under Euclidean constraint [Mit00]. If the weight function obeys the Euclidean metrics in the plane, then, for some restricted variant of the problem, there is a polynomial time algorithm which approximates it within some (large) constant (a corollary of [dBGK⁺02]).

Minimum Watchman Tour and Minimum Watchman Path. The Minimum Watchman Tour (*WT*) and Minimum Watchman Path (*WP*) are the problems of a watchman, who must have a view of n objects, while also trying to minimize the length of the tour (or path). These problems were extensively studied, and given some approximation algorithms as well as solving algorithms for special instances of the problem (i.e., [CN88,NW90,XHHI93,MM95,GN98,CJN99]).

Our Results. We show that G-TSP in 2-D, G-ST in 2-D, WT in 3-D and WP 3-D are all NP-hard to approximate to within any constant factor. This resolves a few open problems presented by Mitchell (see [Mit00], open problems 21, 30 and problem 27 - unconnected part). These problems can be categorized according to three important parameters. One is the dimension of the domain;

the second is whether each subset (region, neighbourhood) is connected and the third is whether sets are pairwise disjoint. For the G-TSP and G-ST problems in 2-D domain our results hold only if sets are allowed to be unconnected (but hold even for pairwise disjoint sets). If each set is connected, (but sets are allowed to coincide) we show an inapproximability factor of $2 - \varepsilon$ for both problems, using an adaptation of a technique from [DBGK⁺02]. In the 3-D domain our results hold for all parameter settings, that is, even when each set is connected and all sets are pairwise disjoint. We also show inapproximability factors of $\frac{\sqrt{k-1}}{\sqrt{2}\sqrt[4]{3}} - \varepsilon$ and $\frac{\sqrt{k-1}}{\sqrt[4]{3}} - \varepsilon$ for the k-G-ST and k-G-TSP, respectively. The following table summarizes the main results for G-TSP and G-ST:

Table 1. Inapproximability factors. The $\forall c$ indicates inapproximability for every constant factor.

G-TSP and G-ST				
Dimension	2-D		3-D or more	
Pairwise Disjoint Sets	Yes	No	Yes	No
connected sets	-	$2 - \varepsilon$	$\forall c$	$\forall c$
unconnected sets	$\forall c$	$\forall c$	$\forall c$	$\forall c$

Outline. We first give some required preliminaries. The first proof shown concerns the approximation hardness factor for G-ST (section 2). The same hardness for G-TSP is then deduced. We next provide the proofs regarding these problems where each region is connected (section 3). The inapproximability factors of WT and WP and the bounded variants k-G-TSP and k-G-ST are given in the full version.

Preliminaries

In order to prove inapproximability of a minimization problem, one usually defines a corresponding gap problem.

Definition 1 (Gap problems). Let A be a minimization problem. *gap-A* $[a, b]$ is the following decision problem:
Given an input instance, decide whether

- there exists a solution of size at most a , or
- every solution of the given instance is of size larger than b .

If the size of the solution resides between these values, then any output suffices.

Clearly, for any minimization problem, if *gap-A* $[a, b]$ is NP-hard, then it is NP-hard to approximate A to within any factor smaller than $\frac{b}{a}$.

Our main result in this paper is derived by a reduction from the vertex-cover in hyper-graph problem. A *hyper-graph* $G = (V, E)$ is a set of vertices V , and a

family E of subsets of V , called edges. It is called k -uniform, if all edges $e \in E$ are of size k , namely $E \subseteq \binom{V}{k}$. A *vertex-cover* of a hyper-graph $G = (V, E)$ is a subset $U \subseteq V$, that "hits" every edge in G , namely, for all $e \in E$, $e \cap U \neq \emptyset$.

Definition 2 (Ek-Vertex-Cover). *The Ek-Vertex-Cover problem is, given a k -uniform graph $G = (V, E)$, to find a minimum size vertex-cover U .*

For $k = 2$ this is the vertex-cover problem on conventional graphs (VC). To prove the approximation hardness result of G-ST (for any constant factor) we use the following approximation hardness of hyper-graph vertex-cover:

Theorem 1. [DGKR03] *For $k > 4$, Gap-Ek-Vertex-Cover- $[\frac{n}{k-1-\varepsilon}, (1-\varepsilon)n]$ is NP-Hard.*

2 Group Steiner Tree and Group TSP in the Plane

Definition 3 (G-ST). *We are given a set P of n points in the plane, and a family X of subsets of P . A solution to the G-ST is a tree T , such that every set $r \in X$ has at least one point in the tree, that is $\forall r \in X, r \cap T \neq \emptyset$. The size (length) of a solution T is the sum of length of all its segments. The objective is to find a solution T of minimal length.*

Let us now prove the main result:

Theorem 2. *G-ST is NP-hard to approximate to within any constant factor.*

Proof. The proof is by reduction from vertex-cover in hyper-graphs to G-ST. The reduction generates an instance X of G-ST, such that the size of its minimal tree T , is related to the size of the minimal vertex-cover U of the input graph $G = (V, E)$. Therefore, an approximation for T would yield an approximation for U , and hence the inapproximability factor known for Gap-Ek-Vertex-Cover, yields an inapproximability factor for G-ST.

The Construction. Given a k -uniform hyper-graph $G = (V, E)$ with $|V| = n$ vertices (assume w.l.o.g. that $\sqrt{n}$ and $\sqrt{\frac{n}{k}}$ are integers), we embed it in the plane to construct a set X of regions. All the regions are subsets of points of a single square of $\sqrt{n} \times \sqrt{n}$ section of the grid. Each point represents an arbitrary vertex of G and each region stands for an edge of G . Formally,

$$P = \{p_{v_i} \mid v_i \in V\}, p_{v_i} = (i \bmod \sqrt{n}, \lfloor \frac{i}{\sqrt{n}} \rfloor)$$

We now define the set of regions X . For every $e \in E$ we define the region r_e to be the union of the k points on the grid, of the vertices in the edge e , namely

$$X = \{r_e \mid e \in E\}, r_e = \{p_v \mid v \in e\}$$

Claim (Soundness). If every vertex cover U of G is of size at least $(1 - \varepsilon)n$ then every solution T for X is of size at least $(1 - \varepsilon)\frac{n}{2}$.

Proof. Inspect a Steiner tree that covers $(1 - \varepsilon)n$ of the grid points. Relate each point on the (segments of the) tree to the nearest covered grid point. As every covered grid point is connected to the tree, the total length related to each covered grid point is at least $\frac{1}{2}$. Thus the size of the tree is at least $\frac{1}{2}(1 - \varepsilon)n$.

Lemma 1 (Completeness). *If there is a vertex cover U of G of size at most $\frac{n}{t}$ then there is a solution T for X of size at most $\frac{3n}{\sqrt{t}}$.*

Proof. We define $T_N(U)$, the natural tree according to a vertex-cover U of G , as follows (see figure 1). A vertical segment on the first column of points, and horizontal segments every d^{th} row, whereas $d = \sqrt{t}$. For every point p_v of $v \in U$ which is not already covered by the tree, we add a segment from it to the closest point q_v on any of the horizontal segments.

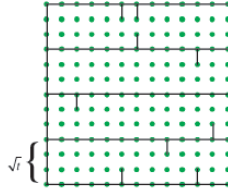


Fig. 1. The Natural Tree

Definition 4 (Natural Tree). *The natural tree $T_N(U)$ of a subset $U \subseteq V$ is the polygon, consisting of the following segments:*

$$T_N(U) = \{((0, 0), (0, \sqrt{n}))\} \cup \{((0, (i-1) \cdot d), (\sqrt{n}, (i-1) \cdot d))\}_{i \in [\frac{\sqrt{n}}{d}]} \cup \{(p_v, q_v)\}_{v \in U}$$

$$q_{v_i} = (i \bmod \sqrt{n}, d \cdot \lfloor \frac{1}{d} \lfloor \frac{i}{\sqrt{n}} \rfloor + \frac{1}{2} \rfloor)$$

Thus, the natural tree contains $\sqrt{\frac{n}{t}} + 1$ horizontal segments of length $\sqrt{n}$ each, a vertical segment of length $\sqrt{n}$ and at most $\frac{n}{t}$ segments, each of length not more than $\frac{\sqrt{t}}{2}$. Therefore

$$|T_N(U)| \leq (\sqrt{\frac{n}{t}} + 2)\sqrt{n} + \frac{n}{t} \cdot \frac{\sqrt{t}}{2} < \frac{3n}{\sqrt{t}}$$

Applying the soundness claim and lemma 1 to theorem 1 we get that for sufficiently large k , Gap-G-ST- $[\frac{3n}{\sqrt{k-1-\varepsilon}}, \frac{1}{2}(1 - \varepsilon)n]$ is NP-Hard. Thus, as k can be arbitrary large, it is NP-hard to approximate G-ST to within any constant factor.

Claim. The approximation hardness factors for G-ST (theorems 2) apply even when regions are pairwise disjoint.

Proof omitted.

Group TSP in the Plane. We next show that the hardness factor shown for G-ST holds for G-TSP with the same parameter settings.

Definition 5 (G-TSP). *In G-TSP in the plane we are given a set P of points in the plane, and a family X of n subsets of P . A solution to the G-TSP is a traversal T , such that every set (region) $r \in X$ has at least one point in the traversal, that is $\forall r \in X, r \cap T \neq \emptyset$. The objective is to find a solution T of minimal length.*

Claim. If there exists an efficient α -approximation for G-TSP then an efficient 2α -approximation for G-ST also exists.

Proof. The size of a minimal tree of G-ST, T_{G-ST}^* , is smaller than the size of a minimal tour of G-TSP, T_{G-TSP}^* , of the same given instance (as one can take a tour and transform it to a tree by removing one of its edges). On the other hand, T_{G-ST}^* is at most $2T_{G-TSP}^*$ (as one can take two copies of the same tree to have a tour). Thus

$$T_{G-ST}^* < T_{G-TSP}^* \leq 2T_{G-ST}^*$$

Therefore, by this claim and theorem 2 we get:

Corollary 1. *G-TSP in the plane is NP-hard to approximate to within any constant factor.*

This holds for the same parameter settings as in G-ST (i.e. unconnected sets for 2-D, and both connected and unconnected for 3-D and higher dimension).

3 The Case of Connected Regions

In the following proofs, we show that G-ST and G-TSP in the plane where *every region is connected* are NP-hard to approximate to within any constant factor smaller than 2. This extends a known factor of $\frac{391}{390} \approx 1.003$ by [DBGK⁺02] for G-TSP. These reductions are similar to theirs but differ in the problem we reduce from (vertex cover on hyper graph instead of vertex cover on graphs of bounded degree) and in some parts of the gadgets constructed.

We consider the 2-D variant of G-TSP, in which each set is connected.

Theorem 3. *G-TSP in the plane with connected regions is NP-hard to approximate to within $2 - \varepsilon$ for any constant $\varepsilon > 0$.*

Proof. Given a hyper-graph $G = (V, E)$ with $|V| = n$ vertices, we construct a set X of regions in the plane. All the regions are subsets of points of two circles in the plane, of perimeter approximately 1. Some of the regions represent edges of G (one region for each edge). Other regions represent vertices of G (l region for each vertex). Let us first describe the set of points of interest P .

The set P is composed of two sets of points, each of which is equally spread on one of two circles. The two circles are concentric, the second one having a slightly larger radius than the first. They are thus referred to as the *inner circle* and the *outer circle*. We will later add to the construction a third circle named the *outmost circle* (see section 3).

P contains a set P_{inner} of nl points on the inner circle ($l = \frac{1}{n\varepsilon}$) and a set P_{outer} of n points on the outer circle, one point for each vertex. We set the radius of the inner circle $\rho \approx \frac{1}{2\pi}$, so that the distance between consecutive points on the inner circle is $\varepsilon = \frac{1}{nl}$. Let us define, formally, the set of points $P = P_{inner} \cup P_{outer}$, which, for the sake of simplicity, would be specified using polar coordinates - namely specifying (radius, angle) of each. Formally,

$$\theta_\varepsilon = \frac{2\pi}{nl} \quad \text{and} \quad \rho = \frac{\frac{\varepsilon}{2}}{\sin(\frac{\theta_\varepsilon}{2})} \approx \frac{1}{2\pi}$$

$$P_{inner} = \{p_{v,j} \mid v \in V, j \in [l]\}, \quad p_{v_i,j} = (\rho, (i \cdot l + j - 1) \cdot \theta_\varepsilon)$$

$$P_{outer} = \{q_v \mid v \in V\}, \quad q_{v_i} = (\rho + \frac{1}{2n}, i \cdot l \cdot \theta_\varepsilon)$$

We now define the set of regions. $X = X_V \cup X_E$ where X_E contains a region for each edge, and X_V contains l regions for each vertex:

$$r_{v,j}^{in} = \{p_{v,j}\} \quad , \quad X_V = \{r_{v,j}^{in} \mid v \in V, j \in [l]\}$$

For every edge $e \in E$ we have a region r_e^{out} composed of points on the outer circle relating to the vertices of e , namely

$$r_e^{out} = \{q_v \mid v \in e\} \quad , \quad X_E = \{r_e^{out} \mid e \in E\}$$

One can easily amend each of the unconnected regions (which are all in X_E) to be connected without changing the correctness of the following proof. For details see last part of this section.

Proof's Idea. We are next going to show, that the most efficient way to traverse X is by traversing all points on the inner circle (say counterclockwise), detouring to visit the closest points on the outer circle, for every point that corresponds to a vertex in the minimal vertex-cover of G (see figure 2).

Definition 6 (Natural Tour). The natural tour $T_N(U)$ of a subset $U \subseteq V$ is the closed polygon, consisting of the following segments:

$$T_N(U) = T_{in} \cup T_{out}$$

$$T_{in} = \{(p_{v,j+1}, p_{v,j+2}) \mid v \in V, j \in [l-2]\} \cup \{(p_{v_i,l}, p_{v_{i+1 \bmod n}, 1}) \mid i \in [n]\}$$

$$T_{out} = \{(p_{v,i}, q_v) \mid v \in U, i \in [2]\}$$

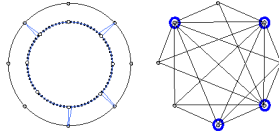


Fig. 2. A vertex-cover and a natural tour

Let us consider the length of this tour $|T_N(U)|$. The natural tour $T_N(U)$ consists of $nl - |U|$ segments of size $\varepsilon = \frac{1}{nl}$ (on the inner circle), $|U|$ segments for the detourings of size $\frac{1}{2n}$ and $|U|$ segments of size in the range $(\frac{1}{2n}, \frac{1}{2n} + \varepsilon)$. Thus $1 + \frac{|U|}{n}(1 - \delta) \leq T_N(U) \leq 1 + \frac{|U|}{n}$ (for some $0 < \delta < \varepsilon$). The exact length of $T_N(U)$ can be computed, but is not important for our purpose. Thus, by the upper bound on $|T_N(U)|$ we have:

Claim. [Completeness] If there is a vertex-cover U of G of size bn , then there is a solution of X of length at most $1 + b$.

Claim. [Soundness] If any vertex-cover U of G is of size at least $a \cdot n$, then any solution of X is of length at least $1 + a - \frac{3}{l}$

Proof. Let T be a solution of X . Clearly T covers all points of P_{inner} (otherwise it is not a solution for X). Let U be a set of vertices that correspond to points on the outer circle, visited by T , namely

$$U = \{v \mid q_v \in T \cap P_{outer}\}$$

Clearly T is a solution only if U is a vertex cover of G , hence $|U| \geq an$. Consider a circle of radius $\frac{1}{2n} - \varepsilon$ around each covered point of the edges regions q_v ($v \in U$). All these circles are pairwise disjoint (as the distance between two points of the edge regions is at least $\frac{1}{n}$). Each one of them contains at least two legs of the path, each of length at least $\frac{1}{2n} - \varepsilon$. In addition the tour visits all the points of the vertex regions, and at least $nl - 3n$ of them are at distance of at least ε from any of the above circles. Thus the in-going path to at least $nl - 3n$ extra points is of length at least ε each. Hence the total length of T is:

$$|T| \geq |U| \cdot 2 \cdot \left(\frac{1}{2n} - \varepsilon\right) + (nl - 3n)\varepsilon \geq a + 1 - \frac{3}{l}$$

Hence by the soundness and completeness claims we have the following:

Lemma 2. *If Gap-Ek-Vertex-Cover[b, a] is NP-hard then for any $\varepsilon > 0$, it is NP-hard to approximate G-TSP in the plane with connected regions, to within $\frac{1+a}{1+b} - \varepsilon$.*

Plugging in the known gap for vertex-cover in hyper-graphs (theorem 1) we get that G-TSP is NP-hard to approximate to within $\frac{1+1-\varepsilon}{1+\frac{1}{k-1-\varepsilon}} - \varepsilon$, hence, for arbitrary small $\varepsilon > 0$ and for a sufficiently large k , G-TSP is NP-hard to approximate to within $2 - \varepsilon$, even if each region is connected.

Making Each Region Connected. To make each region $r_e \in X_E$ connected, we add segments connecting each of the points on the outer circle, to the closest point on a concentric circle (the *outmost circle*, C), of radius $\rho_{outmost}$ suitably large (say, n); namely, for each $q_v \in P_{outer}$, $q_v = (\rho + \frac{1}{2n}, \alpha)$ we add the segment

$$l_v = [(\rho + \frac{1}{2n}, \alpha), (\rho_{outmost}, \alpha)]$$

Edge regions are changed to include the relevant segments and the outmost circle, that is,

$$r_e^{out} = C \cup \bigcup_{v \in e} l_v$$

Vertex regions X_V are left unchanged. Clearly the shortest tour never exits the outer-circle, therefore all points outside the outer-circle may be ignored in the relevant proofs.

Group ST – Connected Regions in the Plane

Theorem 4. *G-ST in the plane with connected regions is NP-hard to approximate to within $2 - \varepsilon$ for any constant $\varepsilon > 0$.*

The proof is very similar to that of G-TSP. For details see full version.

4 Discussion

We have shown that G-TSP, G-ST, WP and WT cannot be efficiently approximated to within any constant factor unless $P = NP$. In this aspect Group-TSP and Group-ST seem to behave more like the Set-Cover problems, rather than the geometric-TSP and geometric-Steiner tree problems.

These reductions illustrate *the importance of gap location*; the approximation hardness result for hyper-graph vertex-cover (see [DGKR03]) is weaker than that of Feige [Fei98], in the sense that the gap ratio is smaller (but works, of course, for the bounded variant). However, their gap location, namely, their almost perfect soundness (see [DGKR03] lemma 4.3), is a powerful tool (see [Pet94]). In the reductions shown here this aspect plays an essential role. We conjecture that the two properties can be joint, namely that,

Conjecture 1. Gap-Hyper-Graph-Vertex-Cover- $[O(\frac{n}{\log n}), (1 - \varepsilon)n]$ is intractable.

Using the exact same reductions, this will extend the known approximation hardness factors of G-TSP, G-ST, WT and WP, as follows:

Corollary 2. *If conjecture 1 is correct then approximating G-TSP in the plane and G-ST in the plane to within $O(\log^{\frac{1}{2}} n)$ is intractable and approximating WT and WP in 3-D to within $O(\log^{\frac{1}{2}} n)$ is also intractable.*

An interesting open problem is whether the square root loss of the approximation hardness factor in the 2-D variant is merely a fault of this reduction or is intrinsic to the plane version of these problems; i.e., is there an approximation with a ratio smaller than $\ln n$ for the plane variants? Are there approximations to the G-TSP and G-ST that perform better in the plane variants than Slavik's [Sla97] approximations for these problems with triangle inequality only? Does higher dimension in these problems impel an increase in complexity? Other open problems remain for the various parameter settings. A most basic variant of G-TSP and G-ST, namely in 2-D domain, where every region is connected, and regions are pairwise disjoint, remains open, as well as the WT and WP on 2-D (open problem 29 of [Mit00]).

Acknowledgments. Many thanks to Shakhbar Smorodinsky, who first brought these problems to our attention; to Matthew J. Katz, for his stimulating lecture on his results; and to Vera Asodi, Guy Kindler and Manor Mendel for their sound advice and insightful comments.

References

- [AH94] E. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 55(3):197–218, 1994.
- [Aro96] S. Arora. Polynomial-time approximation scheme for Euclidean TSP and other geometric problems. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 2–11, 1996.
- [Chr76] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie–Mellon University, 1976.
- [CJN99] S. Carlsson, H. Jonsson, and B. J. Nilsson. Finding the shortest watchman route in a simple polygon. *GEOMETRY: Discrete & Computational Geometry*, 22(3):377–402, 1999.
- [CN88] W. Chin and S. Ntafos. Optimum watchman routes. *Information Processing Letters*, 28(1):39–44, May 1988.
- [CVM00] R. D. Carr, S. Vempala, and J. Mandler. Towards a $4/3$ approximation for the asymmetric traveling salesman problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 116–125, N.Y., January 9–11 2000. ACM Press.
- [dBGK⁺02] M. de Berg, J. Gudmundsson, M. J. Katz, C. Levcopoulos, M. H. Overmars, and A. F. van der Stappen. TSP with neighborhoods of varying size. In *ESA: Annual European Symposium on Algorithms*, pages 187–199, 2002.
- [DGKR03] I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered pcp and the hardness of hypergraph vertex cover. In *Proceedings of the thirty-fifth ACM symposium on Theory of computing*, pages 595–601. ACM Press, 2003.

- [DM01] A. Dumitrescu and J. S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01)*, pages 38–46, New York, January 7–9 2001. ACM Press.
- [EK01] L. Engebretsen and M. Karpinski. Approximation hardness of TSP with bounded metrics. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, pages 201–212, 2001.
- [Fei98] U. Feige. A threshold of $\ln n$ for approximating set cover. *JACM: Journal of the ACM*, 45(4):634–652, 1998.
- [FGM82] A. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric travelling salesman problem. *Networks*, 12:23–39, 1982.
- [GGJ76] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Conference Record of the Eighth Annual ACM Symposium on Theory of Computing*, pages 10–22, Hershey, Pennsylvania, 3–5 May 1976.
- [GGJ77] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32(4):835–859, June 1977.
- [GL99] J. Gudmundsson and C. Levcopoulos. A fast approximation algorithm for TSP with neighborhoods. *Nordic Journal of Computing*, 6(4):469–488, Winter 1999.
- [GN98] L. Gewali and S. C. Ntafos. Watchman routes in the presence of a pair of convex polygons. *Information Sciences*, 105(1-4):123–149, 1998.
- [Ihl92] E. Ihler. The complexity of approximating the class Steiner tree problem. In Gunther Schmidt and Rudolf Berghammer, editors, *Proceedings on Graph-Theoretic Concepts in Computer Science (WG '91)*, volume 570 of *LNCS*, pages 85–96, Berlin, Germany, June 1992. Springer.
- [LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *JACM*, 41(5):960–981, 1994.
- [Mit96] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k -MST problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 402–408, Atlanta, Georgia, 28–30 January 1996.
- [Mit00] J. S. B. Mitchell. *Geometric Shortest Paths and Network optimization*. Elsevier Science, preliminary edition, 2000.
- [MM95] C. Mata and J. S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proceedings of the 11th Annual Symposium on Computational Geometry*, pages 360–369, New York, NY, USA, June 1995. ACM Press.
- [NW90] B. J. Nilsson and D. Wood. Optimum watchmen in spiral polygons. In *CCCG: Canadian Conference in Computational Geometry*, pages 269–272, 1990.
- [Pap77] C. H. Papadimitriou. Euclidean TSP is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.
- [Pet94] E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4(2):133–157, 1994.

- [PV00] C. H. Papadimitriou and S. Vempala. On the approximability of the traveling salesman problem (extended abstract). In ACM, editor, *Proceedings of the thirty second annual ACM Symposium on Theory of Computing: Portland, Oregon, May 21–23, [2000]*, pages 126–133, New York, NY, USA, 2000. ACM Press.
- [RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484. ACM Press, 1997.
- [Sla97] P. Slavik. The errand scheduling problem. Technical Report 97-02, SUNY at Buffalo, March 14, 1997.
- [XHHI93] T. Xue-Hou, T. Hirata, and Y. Inagaki. An incremental algorithm for constructing shortest watchman routes. *International Journal of Computational Geometry and Applications*, 3(4):351–365, 1993.

A Lower Bound for Cake Cutting

Jiří Sgall* and Gerhard J. Woeginger

¹ Mathematical Institute of the Academy of Sciences of the Czech Republic,
Žitná 25, CZ-11567 Praha 1, The Czech Republic. sgall@math.cas.cz

² Department of Mathematics, University of Twente, P.O. Box 217, 7500 AE
Enschede, The Netherlands. g.j.woeginger@math.utwente.nl

Abstract. We prove that in a certain cake cutting model, every fair cake division protocol for n players must use $\Omega(n \log n)$ cuts in the worst case. Up to a small constant factor, our lower bound matches a corresponding upper bound in the same model by Even & Paz from 1984.

1 Introduction

In the cake cutting problem, there are $n \geq 2$ players and a cake $\mathcal{C}$ that is to be divided among the players. Without much loss of generality and in agreement with the cake cutting literature, we will assume throughout the paper that $\mathcal{C} = [0, 1]$ is the unit-interval and the cuts divide the cake into its subintervals. Every player p ($1 \leq p \leq n$) has his own private measure μ_p on sufficiently many subsets of $\mathcal{C}$. These measures μ_p are assumed to be well-behaved; this means that they are:

- Defined on all finite unions of intervals.
- Non-negative: For all $X \subseteq \mathcal{C}$, $\mu_p(X) \geq 0$.
- Additive: For all disjoint subsets $X, X' \subseteq \mathcal{C}$, $\mu_p(X \cup X') = \mu_p(X) + \mu_p(X')$
- Divisible: For all $X \subseteq \mathcal{C}$ and $0 \leq \lambda \leq 1$, there exists $X' \subseteq X$ with $\mu_p(X') = \lambda \cdot \mu_p(X)$.
- Normalized: $\mu_p(\mathcal{C}) = 1$.

All these assumptions are standard assumptions in the cake cutting literature, sometimes subsumed in a concise statement that each μ_p is a probability measure defined on Lebesgue measurable sets and absolutely continuous with respect to Lebesgue measure. We stress that the divisibility of μ_p forbids concentration of the measure in one or more isolated points. As one consequence of this, corresponding open and closed intervals have the same measure, and thus we do not need to be overly formal about the endpoints of intervals.

A cake division *protocol* is an interactive procedure for the players that guides and controls the division process of the cake $\mathcal{C}$. Typically it consists of cut requests like “*Cut cake piece Z into two equal pieces, according to your measure!*” and evaluation queries like “*Is your measure of cake piece Z_1 less, greater, or*

* Partially supported by Institute for Theoretical Computer Science, Prague (project LN00A056 of MŠMT ČR) and grant A1019901 of GA AV ČR.

equal to your measure of cake piece Z_2 ?”). A cake division protocol is not a priori aware of the measures μ_p , but it will learn something about them during its execution. A *strategy* of a player is an adaptive sequence of moves consistent with a given protocol. A cake division protocol is *fair*, if every player p has a strategy that guarantees him a piece of size at least $\mu_p(\mathcal{C})/n$ according to his own measure μ_p . So, even in case $n - 1$ players would all plot up against a single player and would coordinate their moves, then this single player will still be able to get his share of $\mu_p(\mathcal{C})/n$. This is called *simple fair division* in the literature.

In the 1940s, the Polish mathematicians Banach and Knaster designed a simple fair cake division protocol that uses $O(n^2)$ cuts in the worst case; this protocol was explained and discussed in 1948 by Steinhaus [8]. In 1984, Even & Paz [2] used a divide-and-conquer approach to construct a better deterministic protocol that only uses $O(n \log n)$ cuts in the worst case. Remarkably, Even & Paz [2] also design a randomized protocol that uses an expected number of $O(n)$ cuts. For more information on this fair cake cutting problem and on many of its variants, we refer the reader to the books by Brams & Taylor [1] and by Robertson & Webb [7].

The problem of establishing lower bounds for cake cutting goes at least back to Banach (see [8]). Even & Paz [2] explicitly conjecture that there does not exist a fair deterministic protocol with $O(n)$ cuts. Robertson & Webb [7] support and strengthen this conjecture by saying they “would place their money against finding a substantial improvement on the $n \log_2 n$ [upper] bound”.

One basic difficulty in proving lower bounds for cake cutting is that most papers derive upper bound results and to do that, they simply describe a certain procedure that performs certain steps, and then establish certain nice properties for it, but they do not provide a formal definition or a framework. Even & Paz [2] give a proof that for $n \geq 3$, no protocol with $n - 1$ cuts exists; since $n - 1$ cuts are the smallest possible number, such protocols would need to be rather special (in particular they assign a single subinterval to each player) and not much formalism is needed. Only recently, Robertson & Webb [6,7] give a more precise definition of a protocol that covers all the protocols given in the literature. This definition avoids some pathological protocols, but it is still quite general and no super-linear lower bounds are known.

A recent paper [4] by Magdon-Ismail, Busch & Krishnamoorthy proves an $\Omega(n \log n)$ lower bound for a certain non-standard cake cutting model: The lower bound does not hold for the number of performed cuts or evaluation queries, but for the number of comparisons needed to administer these cuts.

Contribution and organization of this paper. We formally define a certain restriction of Robertson-Webb cake cutting model in Section 2. The restrictions are that (i) each player receives a single subinterval of the cake and (ii) the evaluation queries are counted towards the complexity of the protocol together with cuts. Our model is also general enough to cover the $O(n \log n)$ cut deterministic protocol of Even & Paz [2], and we believe that it is fairly natural. We discuss some of the restrictions and drawbacks of our model, and we put it into context with other results from the cake cutting literature. In Section 3 we

then show that in our model, every deterministic fair cake division protocol for n players must use $\Omega(n \log n)$ cuts in the worst case. This result yields the first super-linear lower bound on the number of cuts for simple fair division (in our restricted model), and it also provides a matching lower bound for the result in [2]. Section 4 gives the discussion and open problems.

2 The Restricted Cake Cutting Model

A general assumption in the cake cutting literature is that at the beginning of an execution a protocol has absolutely no knowledge about the measures μ_p , except that they are defined on intervals, non-negative, additive, divisible, and normalized. The protocol issues queries to the players, the players react, the protocols observes their reactions, issues more queries, observes more reactions, and so on, and so on, and so on, and in the end the protocol assigns the cake pieces to the players.

Definition of Robertson-Webb model and our restricted model. We recall that the cake $\mathcal{C}$ is represented by the unit interval. For a real number α with $0 \leq \alpha \leq 1$, the α -point of a player p is the infimum of all numbers x for which $\mu_p([0, x]) = \alpha$ and $\mu_p([x, 1]) = 1 - \alpha$ holds.

In *Robertson-Webb model*, the following two types of queries are allowed.

$\text{Cut}(p; \alpha)$: Player p cuts the cake at his α -point (where $0 \leq \alpha \leq 1$). The value x of the α -point is returned to the protocol.

$\text{Eval}(p; x)$: Player p evaluates the value of the cut x , where x is one of the cuts previously performed by the protocol. The value $\mu_p(x)$ is returned to the protocol.

The protocol can also assign an interval to a player; by doing this several times, a player may end up with a finite union of intervals.

$\text{Assign}(p; x_i, x_j)$: Player p is assigned the interval $[x_i, x_j]$, where $x_i \leq x_j$ are two cuts previously performed by the protocol or 0 or 1.

The complexity of a protocol is given by the number of cuts performed in the worst case, i.e., evaluation queries may be issued for free.

In *our restricted model*, the additional two restrictions are:

$\text{Assign}(p; x_i, x_j)$ is used only once for each p . Hence, in the restricted model every player ends up with a *single* (contiguous) subinterval of the cake.

The complexity of a protocol is given by the number of cuts plus evaluation queries, i.e., each evaluation query contributes to the complexity the same as a cut. Note that this also covers counting only the number of cuts in protocols that do not use evaluation queries at all.

Discussion of the restricted model. The currently best deterministic protocol for exact fair division of Even & Paz [2] does not need evaluation queries and assigns single intervals; we provide a matching bound within these restrictions.

Nevertheless, both restrictions of our model are essential. Protocols in [3,5,6,10], esp. those that achieve not exactly but only approximately fair division, do use evaluation queries, sometimes even a quadratic number of them. The randomized protocol of Even & Paz [2] also uses evaluation queries in addition to expected $O(n)$ cuts; the expected number of evaluation queries is $\Theta(n \log n)$.

We feel that the other restriction, that every player must receive a single, contiguous subinterval of the cake, is perhaps even stronger. By imposing this restriction, it seems that we severely cut down the set of possible protocols; in particular, for some instances, the solution is essentially unique (see our lower bound). Note, however, that *all* known discrete cake cutting protocols from the literature produce solutions where every player ends up with a contiguous subinterval. For instance, all the protocols in [2,3,5,6,8,9,10] have this property. In particular, the divide-and-conquer protocols of Even & Paz [2], both deterministic and randomized, assign single contiguous subinterval to each player, as noted above.

Discussion of Robertson-Webb model. Robertson-Webb model restricts the format of queries to cuts at α points and evaluation queries. This restriction is severe, but it is crucial and essentially unavoidable. Such a restriction must be imposed in one form or the other, just to prevent certain uninteresting types of ‘cheating’ protocols from showing up with a linear number of cuts. Consider the following ‘cheating’ protocol:

(Phase 1).

Every player makes a cut that encodes his i/n -points with $1 \leq i \leq n-1$ (just fix any bijective encoding of $n-1$ real numbers from $[0,1]$ into a single number from $[0,1]$).

(Phase 2).

The protocol executes the Banach-Knaster protocol in the background (Banach-Knaster [8] is a fair protocol that only needs to know the positions of the i/n -points). That is, the protocol determines the relevant cuts without performing them.

(Phase 3).

The protocol tells the players to perform the relevant $n-1$ cuts for the Banach-Knaster solution. If a player does not perform the cut that he announced during the first phase, he is punished and receives an empty piece (and his piece is added to the piece of some other player).

Clearly, every honest player will receive a piece of size at least $1/n$. Clearly, the protocol also works in the friendly environment where every player truthfully executes the orders of the protocol. And clearly, the protocol uses only $2n-1$ cuts—a linear number of cuts. Moreover, there are (straightforward) implementations of this protocol where every player ends up with a single subinterval

of the cake. In cake cutting models that allow announcements of arbitrary real numbers, the cuts in (Phase 1) can be replaced by direct announcements of the i/n -point positions; this yields fair protocols with only $n - 1$ cuts.

These ‘cheating’ protocols are artificial, unnatural and uninteresting, and it is hard to accept them as valid protocols. In Robertson-Webb model they cannot occur, since they violate the form of queries. (One could try to argue that the players might disobey the queries and announce any real number. However, this fails, since the definition of a protocol enforces that a player that honestly answers allowed queries should get a fair share.)

Second important issue is that in the Robertson-Webb model it is sufficient to assume that all players are honest, i.e., execute the commands “Cut at an α -point” and evaluation queries truthfully. Under this assumption all of them get a fair share. Often in the literature, a protocol has no means of enforcing a truthful implementation of these cuts by the players, since the players may cheat, and lie, and try to manipulate the protocol; the requirement is than that any honest player gets a fair share, regardless of the actions of the other players. In Robertson-Webb model, any protocol that works for honest players can be easily modified to the general case as follows. As long as the answers of a player are consistent with some measure, the protocol works with no change, as it assigns a fair share according to this measure (and if the player has a different measure, he lied and has no right to complain). If an inconsistency is revealed (e.g., a violation of non-negativity), the protocol has to be modified to ignore the answers from this player (or rather replace them by some trivial consistent choices).

Of course, in general, the honesty of players is not a restriction on the protocol, but a restriction on the environment. Thus it is of no concern for our lower bound argument which uses only honest players.

In some details our description of the model is different than that of Robertson & Webb. Their formulation in place of evaluation queries is that after performing the cut, its value in all the players’ measures becomes known. This covers all the possible evaluation queries, so it is clearly equivalent if we do not count the number of these queries. However, the number of evaluations may be an interesting parameter, which is why we chose this formulation.

Robertson & Webb also allow cut requests of the form “cut this piece into two pieces with a given ratio of their measures”. This is very useful for an easy formulation of recursive divide-and-conquer protocols. Again, once free evaluation queries are allowed, this is no more general, as we know all the measures of all the existing pieces. Even if we count evaluation queries, we can first evaluate the cuts that created the piece, so such a non-standard cut is replaced by two evaluations and standard cut at some α -point.

Finally, instead cutting at the α -point, Robertson & Webb allow an honest player to return any x with $\mu_p([0, x]) = \alpha$, i.e., we require the answer which is the minimum of the honest answers according to Robertson & Webb. This is a restriction if the instance contains non-trivial intervals of measure zero for some players, otherwise the answer is unique. However, any such instance can

be replaced by a sequence of instances with measures that are very close to the original ones and have non-zero density everywhere. If done carefully, all the α -points in the sequence of modified instances converge to the α -points in the original sequence. Thus the restriction to a particularly chosen honest answer is not essential as well; on the other hand, it keeps the description of our lower bound much simpler.

3 The Proof of the Lower Bound

In this section, we will prove the following theorem by means of an adversary argument in a decision tree.

Theorem 1. *In the restricted cake cutting model of Section 2 (where each player is assigned a single interval), every deterministic fair cake division protocol for n players uses at least $\Omega(n \log n)$ cuts and/or evaluation queries in the worst case.*

The adversary continuously observes the actions of the deterministic protocol, and he reacts by fixing the measures of the players appropriately.

Let us start by describing the specific cake measures μ_p that we use in the input instances. Let $\varepsilon < 1/n^4$ be some small, positive real number. For $i = 1, \dots, n$ we denote by $X_i \subset [0, 1]$ the set consisting of the n points $i/(n+1) + k \cdot \varepsilon$ with $1 \leq k \leq n$. Moreover, we let $X = \bigcup_{0 \leq i \leq n} X_i$. For $p = 1, \dots, n$, by definition the player p has his 0-point at position 0. The positions of the i/n -points with $1 \leq i \leq n$ are fixed by the adversary during the execution of the protocol: The i/n -points of all players are taken from X_i , and distinct players receive distinct i/n -points. As one consequence, all the i/n -points of all players will lie strictly to the left of all the $(i+1)/n$ -points of all players.

All the cake value for player p is concentrated in tiny intervals $\mathcal{I}_{p,i}$ of length ε that are centered around his i/n -points: For $i = 0, \dots, n$, the measure of player p has a sharp peak with value $i/(n^2 + n)$ immediately to the left of his i/n -point and a sharp peak with value $(n - i)/(n^2 + n)$ immediately to the right of his i/n -point. Note that the measure between the i/n -point and the $(i+1)/n$ -point indeed adds up to $1/n$. Moreover, the measures of the two peaks around every i/n -point add up to $1/(n+1)$, and the intervals that support these peaks for different players are always disjoint, with the exception of the intervals $\mathcal{I}_{p,0}$ that are the same for all the players. We do not explicitly describe the shape of the peaks; it can be arbitrary, but determined in advance and the same for each player.

For every player p , the portions of the cake between interval $\mathcal{I}_{p,i}$ and interval $\mathcal{I}_{p,i+1}$ have measure 0 and hence are worthless to p . By our definition of α -points, every α -point of player p will fall into one of his intervals $\mathcal{I}_{p,i}$ with $0 \leq i \leq n$. If a player p cuts the cake at some point $x \in \mathcal{I}_{p,i}$, then we denote by $c_p(x)$ the corresponding i/n -point of player p .

Lemma 1. *Let x be a cut that was done by player s , and let $y \geq x$ be another cut that was done by player t . Let $\mathcal{J} = [x, y]$ and $\mathcal{J}' = [c_s(x), c_t(y)]$. If $\mu_p(\mathcal{J}) \geq 1/n$ holds for some player p , then also $\mu_p(\mathcal{J}') \geq 1/n$.*

Proof. (Case 1) If $s = p$ and $t = p$, then let $\mathcal{I}_{p,j}$ and $\mathcal{I}_{p,k}$ be the intervals that contain the points $c_p(x)$ and $c_p(y)$, respectively. Then $\mu_p(\mathcal{J}) \geq 1/n$ implies $k \geq j + 1$. The measure $\mu_p(\mathcal{J}')$ is at least the measure $(n - j)/(n^2 + n)$ of the peak immediately to the right of the j/n -point plus the measure $k/(n^2 + n)$ immediately to the left of the k/n -point, and these two values add up to at least $1/n$.

(Case 2) If $s = p$ and $t \neq p$, then let $\mathcal{I}_{p,j}$ be the interval that contains $c_p(x)$. Then $\mu_p(\mathcal{J}) \geq 1/n$ implies that $\mathcal{J}$ and $\mathcal{J}'$ both contain $\mathcal{I}_{p,j+1}$, and again $\mu_p(\mathcal{J}')$ is at least $1/n$. Note that the argument works also if $j = 0$.

(Case 3) The case $s \neq p$ and $t = p$ is symmetric to the second case above.

(Case 4) If $s \neq p$ and $t \neq p$, then the interval between x and $c_s(x)$ and the interval between y and $c_t(y)$ both have measure 0 for player p . By moving these two cuts, we do not change the value of $\mathcal{J}$ for p . $\square$

We call a protocol *primitive*, if in all of its cut operations $\text{Cut}(p; \alpha)$ the value α is of the form i/n with $0 \leq i \leq n$.

Lemma 2. *For every protocol $\mathcal{P}$ in the restricted model, there exists a primitive protocol $\mathcal{P}'$ in the restricted model, such that for every cake cutting instance I of the restricted form described above,*

- $\mathcal{P}$ and $\mathcal{P}'$ make the same number of cuts on I ,
- if $\mathcal{P}$ applied to instance I assigns to player p a piece $\mathcal{J}$ of measure $\mu_p(\mathcal{J}) \geq 1/n$, then also $\mathcal{P}'$ applied to instance I assigns to player p a piece $\mathcal{J}'$ of measure $\mu_p(\mathcal{J}') \geq 1/n$.

Proof. Protocol $\mathcal{P}'$ imitates protocol $\mathcal{P}$. Whenever $\mathcal{P}$ requests player p to cut at his α -point x with $0 < \alpha < 1$, then $\mathcal{P}'$ computes the unique integer k with

$$\frac{k}{n+1} < \alpha \leq \frac{k+1}{n+1}$$

Then $\mathcal{P}'$ requests player p to cut the cake at his k/n -point. Note that by the choice of k , this k/n -point equals $c_p(x)$. The value of the cuts at x and $c_p(x)$ is the same for all the players other than p , thus any following answer to an evaluation query is the same in $\mathcal{P}'$ and $\mathcal{P}$. Furthermore, since the shape of the peaks is predetermined and the same for all the players, from the cut of $\mathcal{P}'$ at $c_p(x)$ we can determine the original cut of $\mathcal{P}$ at x . Consequently $\mathcal{P}'$ can simulate all the decisions of $\mathcal{P}$. When assigning pieces, each original cut x of $\mathcal{P}$ is replaced by the corresponding cut $c_p(x)$ of $\mathcal{P}'$. Clearly, both protocols make the same number of cuts, and Lemma 1 yields that if $\mathcal{P}$ is fair, then also $\mathcal{P}'$ is fair. $\square$

Hence, from now on we may concentrate on some fixed primitive protocol $\mathcal{P}^*$, and on the situation where all cuts are from the set X . The strategy of the

adversary is based on a permutation π of the integers $1, \dots, n$; this permutation π is kept secret and not known to the protocol $\mathcal{P}^*$.

Now assume that at some point in time protocol $\mathcal{P}^*$ asks player p to perform a cut at his i/n -point. Then the adversary fixes the measures as follows:

- If $\pi(p) < i$, then the adversary assigns the i/n -point of player p to the smallest point in the set X_i that has not been used before.
- If $\pi(p) > i$, then the adversary assigns the i/n -point of player p to the largest point in the set X_i that has not been used before.
- If $\pi(p) = i$, then the adversary assigns the i/n -point of player p to the i th smallest point in the set X_i .

Consequently, any possible assignment of i/n -points to points in X_i has the following form: The player q with $\pi(q) = i$ sits at the i th smallest point. The $i - 1$ players with $\pi(p) \leq i - 1$ are at the first (smallest) $i - 1$ points, and the $n - i$ players with $\pi(p) \geq i + 1$ are at the last (largest) $n - i$ points. The precise ordering within the first $i - 1$ and within the last $n - i$ players depends on the behavior of the protocol $\mathcal{P}^*$. When protocol $\mathcal{P}^*$ terminates, then the adversary fixes the ordering of the remaining i/n -points arbitrarily (but in agreement with the above rules).

Lemma 3. *If $\pi(p) \leq i \leq \pi(q)$ and $p \neq q$, then in the ordering fixed by the adversary the i/n -point of player p strictly precedes the i/n -point of player q .*

Proof. Immediately follows from the adversary strategy above. $\square$

If the protocol $\mathcal{P}^*$ asks a player p an evaluation query on an existing cut at i/n -point of player p' , the current assignment of i/n -points to points in X_i and the permutation π determine if the i/n -point of player p is smaller or larger than that of p' (for all the possible resulting assignment obeying the rules above). This is all that is necessary to determine the value of the cut, and thus the adversary can generate an honest answer to the query.

At the end, the primitive protocol $\mathcal{P}^*$ must assign intervals to players: $\mathcal{P}^*$ selects $n - 1$ of the performed cuts, say the cuts at positions $0 \leq y_1 \leq y_2 \leq \dots \leq y_{n-1} \leq 1$; moreover, we define $y_0 = 0$ and $y_n = 1$. Then for $i = 1, \dots, n$, the interval $[y_{i-1}, y_i]$ goes to player $\phi(i)$, where ϕ is a permutation of $1, \dots, n$.

Lemma 4. *If the primitive protocol $\mathcal{P}^*$ is fair, then*

- (a) $y_i \in X_i$ holds for $1 \leq i \leq n - 1$.
- (b) The interval $[y_{i-1}, y_i]$ contains the $(i - 1)/n$ -point and the i/n -point of player $\phi(i)$, for every $1 \leq i \leq n$.

Proof. (a) If y_1 is at an $0/n$ -point of some player, then $y_1 = 0$ and piece $[y_0, y_1]$ has measure 0 for player $\phi(1)$. If $y_{n-1} \in X_n$, then piece $[y_{n-1}, y_n]$ has measure at most $1/(n + 1)$ for player $\phi(n)$. If $y_{i-1} \in X_j$ and $y_i \in X_j$ for some $2 \leq i \leq n - 1$ and $1 \leq j \leq n - 1$, then player $\phi(i)$ receives the piece $[y_{i-1}, y_i]$ of measure at most $1/(n + 1)$. This leaves the claimed situation as the only possibility.

(b) Player $\phi(i)$ receives the cake interval $[y_{i-1}, y_i]$. By the statement in (a), this interval can not cover player $\phi(i)$'s measure-peaks around j/n -points with $j < i - 1$ or with $j > i$. The two peaks around the $(i - 1)/n$ -point of player $\phi(i)$ yield only a measure of $1/(n + 1)$; thus the interval cannot avoid the i/n -point. A symmetric argument shows that the interval cannot avoid the $(i - 1)/n$ -point of player $\phi(i)$. $\square$

Lemma 5. *For any permutation $\sigma \neq \text{id}$ of the numbers $1 \dots n$, there exists some $1 \leq i \leq n$ with $\sigma(i + 1) \leq i \leq \sigma(i)$.*

Proof. Take the minimum i with $\sigma(i + 1) \leq i$. $\square$

Finally, we claim that $\phi = \pi^{-1}$. Suppose otherwise. Then $\pi \circ \phi \neq \text{id}$ and by Lemma 5 there exists an i such that

$$\pi(\phi(i + 1)) \leq i \leq \pi(\phi(i)).$$

Let $p := \phi(i + 1)$ and $q := \phi(i)$, let z_p denote the i/n -point of player p , and let z_q denote the i/n -point of player q . Lemma 3 yields $z_p < z_q$. According to Lemma 4.(b), point z_p must be contained in $[y_i, y_{i+1}]$ and point z_q must be contained in $[y_{i-1}, y_i]$. But this implies $z_p \geq y_i \geq z_q$ and blatantly contradicts $z_p < z_q$.

This contradiction shows that the assignment permutation ρ of protocol $\mathcal{P}^*$ must be equal to the inverse permutation of π . Hence, for each permutation π the primitive protocol must reach a different leaf in the underlying decision tree. After an evaluation query $\text{Eval}(p; x)$, where x is a result of $\text{Cut}(p'; i/n)$, for $p \neq p'$ and $1 \leq i < n$, the protocol is returned one of only two possible answers, namely $i/(n + 1)$ or $(i + 1)/(n + 1)$, indicating if $\text{Cut}(p; i/n)$ is before or after x in X_i (if $p = p'$ or $i \in \{0, n\}$, the answer is unique and trivial). After every query $\text{Cut}(p; i/n)$, the primitive protocol is returned one point of X_i : namely the first unused point if $\pi(p) < i$, the last unused point if $\pi(p) > i$, or the i th point if $\pi(p) = i$. Since the values in X_i are known in advance, the whole protocol can be represented by a tree with a binary node for each possible evaluation query and a ternary node for each possible cut. The depth of a leaf in the tree is the number of cuts and evaluation queries performed for an instance corresponding to a given permutation. Since there are $n!$ permutations, the maximal depth of a leaf corresponding to some permutation must be at least $\log_3(n!) = \Omega(n \log n)$. This completes the proof of Theorem 1.

4 Discussion

One contribution of this paper is a discussion of various models and assumptions for cake cutting (that appeared in the literature in some concise and implicit form) and a definition of a restricted model that covers the best protocols known.

The main result is a lower bound of $\Omega(n \log n)$ on the number of cuts and evaluation queries needed for simple fair division in this restricted n -player cake

cutting model. The model clearly has its weak points (see, again, the discussion in Section 2), and it would be interesting to provide similar bounds in less restricted models. In particular, we suggest the two open problems, related to the two restrictions in our model.

Assigning More Subintervals

Problem 1. How many cuts are needed if no evaluation queries are allowed (but any player can be assigned several intervals)?

Our lower bound argument seems to break down even for ‘slight’ relaxations of the assumption about a single interval: On the instances from our lower bound, one can easily in $O(n)$ cuts assign to each player two of the intervals of size ε that support his measure and this is clearly sufficient. And we do not even know how to make the lower bound work for the case where the cake is a circle, that is, for the cake that results from identifying the points 0 and 1 in the unit interval or equivalently when a single player can receive a share of two intervals, one containing 0 and one containing 1. (Anyway, the circle is considered a non-standard cake and is not treated anywhere in the classical cake cutting literature [1,7].)

The restriction to a single subinterval share for each player seems very significant in our lower bound technique. On the other hand, all the protocols known to us obey this restriction.

Evaluation Queries

Problem 2. How many cuts are needed if any player is required to receive a single subinterval (but evaluation queries are allowed and free)?

With evaluation queries, our lower bound breaks, since the decision tree is no longer ternary. After performing a cut, we may learn that $\pi(p) < i$ or $\pi(p) > i$, in which case we gain no additional information. However, once we find i such that $\pi(p) = i$, the protocol finds out all values of p' satisfying $\pi(p') < i$ and we can recurse on the two subinstances. We can use this to give a protocol that uses only $O(n \log \log n)$ cuts (and free evaluation queries) and works on the instances from our lower bound.

The currently best deterministic protocol for exact fair division of Even & Paz [2] does not need evaluation queries. However, other protocols in [3,5,6,10], in particular those that achieve not exactly but only approximately fair division, do use evaluation queries. Also the randomized protocol of Even & Paz [2] with expected $O(n)$ cuts uses expected $\Theta(n \log n)$ evaluation queries. Thus it would be very desirable to prove a lower bound for a model including free evaluation queries, or perhaps find some trade-off between cuts and evaluation queries.

The protocols actually use only limited evaluations like “Is your measure of cake piece Z less, greater, or equal to the threshold τ ?” or “Is your measure of cake piece Z_1 less, greater, or equal to your measure of cake piece Z_2 ?”. Perhaps handling these at first would be more accessible. We hope that this problem

could be attacked by a similar lower bound technique using the decision trees in connection with a combinatorially richer set of instances.

Another interesting question concerns the randomized protocols. The randomized protocol of Even & Paz [2] uses an expected number of $O(n)$ cuts and $\Theta(n \log n)$ evaluation queries. Can the number of evaluation queries be decreased? Or can our lower bound be extended to randomized protocols?

Finally, let us remark that our model seems to be incomparable with that of Magdon-Ismael, Busch & Krishnamoorthy [4]. The set of instances for which they prove a lower bound of $\Omega(n \log n)$ on the number of comparisons can be easily solved with $O(n)$ cuts with no evaluation queries even in our restricted model. On the other hand, they prove a lower bound for protocols that have no restriction similar to our requirement of assigning a single subinterval to each player. The common feature of both models seems to be exactly the lack of ability to incorporate the free evaluation queries; note that using an evaluation query generates at least one comparison.

Acknowledgment. We thank anonymous referees for providing several comments that helped us to improve the paper.

References

1. S.J. BRAMS AND A.D. TAYLOR (1996). *Fair Division – From cake cutting to dispute resolution*. Cambridge University Press, Cambridge.
2. S. EVEN AND A. PAZ (1984). A note on cake cutting. *Discrete Applied Mathematics* 7, 285–296.
3. S.O. KRUMKE, M. LIPMANN, W. DE PAEPE, D. POENSGEN, J. RAMBAU, L. STOUGIE, AND G.J. WOEGINGER (2002). How to cut a cake almost fairly. *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2002)*, 263–264.
4. M. MAGDON-ISMAIL, C. BUSCH, AND M.S. KRISHNAMOORTHY (2003). Cake cutting is not a piece of cake. *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2003)*, LNCS 2607, Springer Verlag, 596–607.
5. J.M. ROBERTSON AND W.A. WEBB (1991). Minimal number of cuts for fair division. *Ars Combinatoria* 31, 191–197.
6. J.M. ROBERTSON AND W.A. WEBB (1995). Approximating fair division with a limited number of cuts. *Journal of Combinatorial Theory, Series A* 72, 340–344.
7. J.M. ROBERTSON AND W.A. WEBB (1998). *Cake-cutting algorithms: Be fair if you can*. A.K. Peters Ltd.
8. H. STEINHAUS (1948). The problem of fair division. *Econometrica* 16, 101–104.
9. W.A. WEBB (1997). How to cut a cake fairly using a minimal number of cuts. *Discrete Applied Mathematics* 74, 183–190.
10. G.J. WOEGINGER (2002). An approximation scheme for cake division with a linear number of cuts. *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'2002)*, LNCS 2461, Springer Verlag, 896–901.

Ray Shooting and Stone Throwing^{*}

Micha Sharir¹ and Hayim Shaul²

¹ School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel and
Courant Institute of Mathematical Sciences, New York University, New York, NY
10012, USA. michas@post.tau.ac.il

² School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel.
hayim@post.tau.ac.il

Abstract. The paper presents two algorithms involving shooting in three dimensions. We first present a new algorithm for performing ray shooting amid several special classes of n triangles in three dimensions. We show how to implement this technique to obtain improved query time for a set of fat triangles, and for a set of triangles stabbed by a common line. In both cases our technique requires near-linear preprocessing and storage, and answers a query in about $n^{2/3}$ time. This improves the best known result of close to $n^{3/4}$ query time for general triangles. The second algorithm handles stone-throwing amid arbitrary triangles in 3-space, where the curves along which we shoot are vertical parabolas, which are trajectories of stones thrown under gravity. We present an algorithm that answers stone-throwing queries in about $n^{3/4}$ time, using near linear storage and preprocessing. As far as we know, this is the first nontrivial solution of this problem. Several extensions of both algorithms are also presented.

1 Introduction

The ray shooting problem is to preprocess a set of objects such that the first object hit by a query ray can be determined efficiently. The ray shooting problem has received considerable attention in the past because of its applications in computer graphics and other geometric problems. The planar case has been studied thoroughly. Optimal solutions, which answer a ray shooting query in $O(\log n)$ time using $O(n)$ space, have been proposed for some special cases [8,10]. For an arbitrary collection of segments in the plane, the best known algorithm answers a ray shooting query in time $O(\frac{n}{\sqrt{s}} \log^{O(1)} n)$ using $O(s^{1+\epsilon})$ space and preprocessing [3,6], for any $\epsilon > 0$, where s is a parameter that can vary between n and n^2 (we follow the convention that bounds that depend on ϵ hold for any

^{*} This work was supported by a grant from the Israel Science Fund (for a Center of Excellence in Geometric Computing), and is part of the second author's Ph.D. dissertation, prepared under the supervision of the first author in Tel Aviv University. Work by Micha Sharir was also supported by NSF Grants CCR-97-32101 and CCR-00-98246, by a grant from the U.S.-Israel Binational Science Foundation, and by the Hermann Minkowski-MINERVA Center for Geometry at Tel Aviv University.

$\epsilon > 0$, where the constant of proportionality depends on ϵ , and generally tends to ∞ as ϵ tends to 0).

The three-dimensional ray shooting problem seems much harder and it is still far from being fully solved. Most studies of this problem consider the case where the given set is a collection of triangles. If these triangles are the faces of a convex polyhedron, then an optimal algorithm can be obtained using the hierarchical decomposition scheme of Dobkin and Kirkpatrick [12]. If the triangles form a polyhedral terrain (an xy -monotone piecewise-linear surface), then the technique of Chazelle et al. [9] yields an algorithm that requires $O(n^{2+\epsilon})$ space and answers ray shooting queries in $O(\log n)$ time. The best known algorithm for the general ray shooting problem (involving triangles) is due to Agarwal and Matoušek [5]; it answers a ray shooting query in time $O(\frac{n^{1+\epsilon}}{s^{1/4}})$, with $O(s^{1+\epsilon})$ space and preprocessing. The parameter s can range between n and n^4 . See [1, 5] for more details. A variant of this technique was presented in [7] for the case of ray shooting amid a collection of convex polyhedra.

On the other hand, there are certain special cases of the 3-dimensional ray shooting problem which can be solved more efficiently. For example, if the objects are planes or halfplanes, ray shooting amid them can be performed in time $O(\frac{n^{1+\epsilon}}{s^{1/3}})$, with $O(s^{1+\epsilon})$ space and preprocessing; see [4] for details. If the objects are horizontal fat triangles or axis-parallel polyhedra, ray shooting can be performed in time $O(\log n)$ using $O(n^{2+\epsilon})$ space; see [11] for details. If the objects are spheres, ray shooting can be performed in time $O(\log^4 n)$ with $O(n^{3+\epsilon})$ space; see [15].

In this paper we consider two special cases of the ray shooting problem: the case of arbitrary fat triangles and the case of triangles stabbed by a common line. We present an improved solution for the case where only near-linear storage is allowed. Specifically, we improve the query time to $O(n^{2/3+\epsilon})$, using $O(n^{1+\epsilon})$ space and preprocessing. Curiously, at the other end of the trade-off, we did not manage to improve upon the general case, and so $O(n^{4+\epsilon})$ storage is still required for logarithmic-time queries. These two extreme bounds lead to a different trade-off, which is also presented in this paper.

Next we study another problem of shooting along arcs amid triangles in three dimensions, which we refer to as *stone throwing*. In this problem we are given a set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$, and we wish to preprocess them into a data structure that can answer efficiently stone throwing queries, where each query specifies a point $p \in \mathbb{R}^3$ and a velocity vector $\mathbf{v} \in \mathbb{R}^3$; these parameters define a vertical parabolic trajectory traced by a stone thrown from p with initial velocity $\mathbf{v}$ under gravity (which we assume to be exerted in the negative z -direction), and the query asks for the first triangle of $\mathcal{T}$ to be hit by this trajectory. The query has six degrees of freedom, but the parabola π that contains the stone trajectory has only five degrees of freedom, which is one more than the number of degrees of freedom for lines in space.

Unlike the case of ray shooting, we consider here the basic case where the triangles of $\mathcal{T}$ are arbitrary, and present a solution that uses near-linear storage and answers stone-throwing queries in time near $n^{3/4}$. These bounds are

interesting, since they are identical to the best bounds known for the general ray-shooting problem, even though the stone-throwing problem appears to be harder since it involves one additional degree of freedom. At present we do not know whether the problem admits a faster solution for the special classes of triangles considered in the first part of the paper. Moreover, at the other extreme end of the trade-off, where we wish to answer stone-throwing queries in $O(\log n)$ time, the best solution that we have requires storage near n^5 , which is larger, by a factor of n , than the best known solution for the ray-shooting problem. (This latter solution is omitted in this abstract.)

As far as we know, this is the first non-trivial treatment of the stone throwing problem. The method can be easily extended to answer shooting queries along other types of trajectories, with similar performance bounds (i.e., near linear storage and preprocessing and near $n^{3/4}$ query time). In fact, this holds for shooting along the graph of any univariate algebraic function of constant degree that lies in any vertical plane.

2 Ray Shooting amid Fat Triangles and Related Cases

2.1 Preliminaries

In this paper we assume that the triangles are not “too vertical”. More formally, we assume that the angle formed between the xy -plane and the plane that supports any triangle in $\mathcal{T}$ is at most θ , where $\cos \theta = \frac{1}{\sqrt{3}}$. Steeper triangles can (and will) be handled by an appropriate permutation of the coordinate axes.

A triangle Δ is α -fat (or *fat*, in short) if all its internal angles are bigger than some fixed angle α .

A *positive curtain* (resp., *negative curtain*) is an unbounded polygon in space with three edges, two of which are parallel to the z -axis and extend to $z = +\infty$ (resp., $z = -\infty$). In the extreme case where these vertical edges are missing, the curtain is a vertical halfplane bounded by a single line. Curtains have been studied in [11], but not as an aid for a general ray shooting problem, as studied here.

Given a segment s in space, we denote by $C^+(s)$ (resp., $C^-(s)$) the positive (resp., negative) curtain that is defined by s , i.e., whose bounded edge is s .

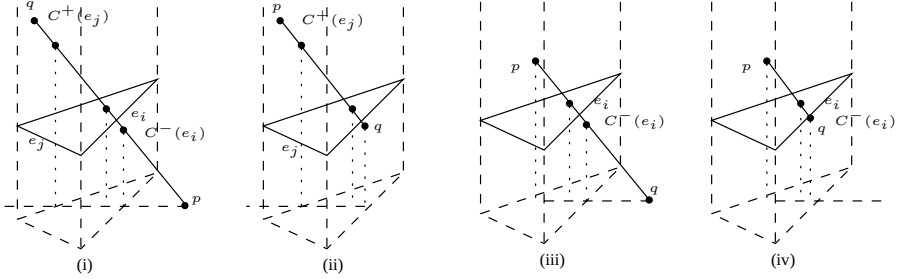
We say that a point p is above (below) a triangle Δ if the vertical projection of p on the xy -plane lies inside the vertical projection of Δ , and p is above (below) the plane containing Δ . The next two lemmas are trivial.

Lemma 1. *Given a non-vertical triangle Δ with three edges e_1, e_2 and e_3 and a segment $\overline{pq}$, all in $\mathbb{R}^3$, then the segment $\overline{pq}$ intersects the triangle Δ if and only if (exactly) one of the following conditions holds:*

- (i) $\overline{pq}$ intersects one positive curtain $C^+(e_i)$ and one negative curtain $C^-(e_j)$ of two edges e_i, e_j of Δ .
- (ii) One of the endpoints is below Δ and $\overline{pq}$ intersects one positive curtain $C^+(e_i)$ of Δ .

- (iii) One of the endpoints is above Δ and $\overline{pq}$ intersects one negative curtain $C^-(e_i)$ of Δ .
- (iv) p is above Δ and q is below Δ or vice versa.

Proof. Straightforward; see the figure.



Lemma 2. Given a segment $\overline{p_1p_2}$, contained in a line ℓ_1 , and two points s_1 and s_2 , contained in a line ℓ_2 , all in the plane, then the segment $\overline{p_1p_2}$ intersects the segment $\overline{s_1s_2}$ if and only if both of the following conditions hold: (i) s_1 and s_2 are on different sides of ℓ_1 , and (ii) p_1 and p_2 are on different sides of ℓ_2 .

2.2 Overview of the Algorithm

We first sketch the outline of our algorithm before describing it in detail.

Reduction to segment emptiness. Given a set $\mathcal{T}$ of fat triangles (the case of triangles stabbed by a common line is similar) and a ray ρ , specified by a point p and a direction d , we want to determine the first triangle $t^* \in \mathcal{T}$ intersected by ρ . We use the parametric search technique, as in Agarwal and Matoušek [4], to reduce this problem to the segment emptiness problem, that is, to the problem of determining whether a query segment $\overline{pq}$ intersects any triangle in $\mathcal{T}$.

An algorithm that solves this latter segment-emptiness problem proceeds through the following steps.

Partitioning a triangle into semi-canonical triangles. We start by decomposing each triangle into four sub-triangles, where each sub-triangle has two edges parallel to two planes in some predefined set $\mathcal{D}$ of $O(1)$ planes. This decomposition is facilitated by the fact that the given triangles are α -fat, and the size of the set $\mathcal{D}$ depends on α . We divide the $4n$ resulting triangles into $|\mathcal{D}|^2$ sets, where all triangles in the same set have two edges parallel to the same pair of planes, and we act on each set separately. Assume without loss of generality that we are given a set of triangles with one edge parallel to the xz -plane and one edge parallel to the yz -plane (this can always be enforced using an appropriate affine transformation). We refer to this pair of edges as *semi-canonical*.

Discarding triangles that do not intersect the xy -projection of the query segment. We project the triangles and the query segment $\overline{pq}$ on the xy -plane, and obtain a compact representation of the set of all triangles whose xy -projections are intersected by the projection of $\overline{pq}$. This set will be the union of a small number of canonical, preprocessed sets, and we apply the subsequent stages of the algorithm to each such subset. Moreover, we construct these sets in such a way that allows us to know which pair of edges of each triangle t in a canonical set are intersected by the segment in the projection. At least one of these edges is necessarily semi-canonical; we call it e_c^t , and call the other edge e_r^t .

Checking for intersection with curtains. We next need to test whether there exists a triangle t^* in a given canonical set such that the query segment intersects the positive curtain $C^+(e_c^t)$ and the negative curtain $C^-(e_r^t)$. The symmetric case, involving $C^-(e_c^t)$ and $C^+(e_r^t)$, is handled similarly. Checking the other conditions for $\overline{pq}$ intersecting t , namely, checking whether p is below (resp., above) t and $\overline{pq}$ intersects some positive curtain $C^+(e)$ (resp., negative curtain $C^-(e)$) of t , or whether p lies below t and q lies above t , is simpler and we skip these steps in this abstract.

We first collect all triangles t in a canonical subset so that $\overline{pq}$ intersects the positive curtain $C^+(e_c^t)$ erected from the semi-canonical edge e_c^t of t . The output is again a union of a small number of canonical sets. The fact that the edges e_c^t are semi-canonical allows us to represent these curtains as points in a 3-dimensional space (rather than 4-dimensional as in the case of general curtains or lines in space), and this property is crucial for obtaining the improved query time.

Finally, for each of the new canonical sets, we test whether the segment intersects the negative curtain $C^-(e_r^t)$, erected over the other edge e_r^t of at least one triangle in the set. This step is done using the (standard) representation of lines in $\mathbb{R}^3$ as points or halfplanes in the 5-dimensional Plücker space, and exploiting the linear structure of this representation [9]. Symmetrically, we test over all canonical sets, whether $\overline{pq}$ intersects the negative curtain $C^-(e_c^t)$ and the positive curtain $C^+(e_r^t)$ of at least one triangle t . Any successful test at this stage implies that $\overline{pq}$ intersects a triangle in $\mathcal{T}$, and vice versa.

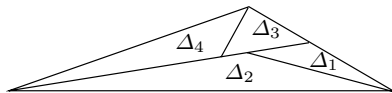
As our analysis will show, this multi-level structure uses $O(n^{1+\epsilon})$ space and preprocessing and can answer queries in $O(n^{2/3+\epsilon})$ time, for any $\epsilon > 0$.

2.3 Partitioning a Triangle into Semi-canonical Triangles

Recall that we assume that the triangles are not “too vertical”. There exists a set of vertical planes D of size $O(1/\alpha)$, such that, for each vertex v of any α -fat triangle t which is not too vertical, it is possible to split t into two (non-empty) triangles by a segment incident to v which is parallel to some plane in D . We say that this segment is *semi-canonical*.

Given a set $\mathcal{T}$ of such α -fat, not-too-vertical triangles, we decompose each triangle $\Delta \in \mathcal{T}$ into four triangles, such that each new triangle has at least two

semi-canonical edges. This is easy to do, in the manner illustrated in the figure. We refer to the resulting sub-triangles as *semi-canonical*.



We partition $\mathcal{T}$ into $O(1/\alpha^2)$ canonical families, where all triangles in the same family have two edges parallel to two fixed canonical planes. We preprocess each family separately. Let $\mathcal{F}$ be a fixed canonical family. Let us assume, without loss of generality, that the two corresponding canonical planes are the xz -plane and the yz -plane. (Since our problem is affine-invariant, we can achieve this reduction by an appropriate affine transformation.)

2.4 Finding Intersections in the xy -Projection

Project the triangles in $\mathcal{F}$ and the segment $\overline{pq}$ onto the xy -plane. Denote the projection of an object a by $\underline{a}$. We need to compute the set of all triangles whose xy -projections are crossed by $\underline{pq}$ and associate with each output triangle Δ two edges of Δ whose projections are crossed by $\underline{pq}$. The output will be represented as the union of canonical precomputed sets.

For lack of space we omit the description of this step. It is fully standard, and is applied in the previous ray-shooting algorithms (see, e.g., [2]). It combines several levels of two-dimensional range searching structures, using Lemma 2 to check for intersections between $\underline{pq}$ and the projected triangle edges.

2.5 Finding Intersections with Semi-Canonical Curtains

Let $\mathcal{T}'$ be one of the canonical sets output in the previous stage, and for each $t \in \mathcal{T}'$, let e_c^t, e_r^t denote the two edges of t whose xy -projections are crossed by the projection of the query segment $\overline{pq}$, where e_c^t is semi-canonical. In the next stage we preprocess $\mathcal{T}$ into additional levels of the data structure, which allows us to compute the subset of all those triangles for which $\overline{pq}$ intersects $C^-(e_c^t)$.

Recall that we apply an affine transformation such that each e_c^t is parallel to the xz -plane or to the yz -plane. Let us assume, without loss of generality, that all e_c^t are parallel to the xz -plane. Since we know that the projection of $\overline{pq}$ intersects each e_c^t and e_r^t , we can extend the query segment and the edges e_c^t, e_r^t to full lines.

The extended negative curtain $C^-(e_c^t)$ has three degrees of freedom, and can be represented as $C_{\zeta, \eta, \xi} = \{(x, y, z) | y = \xi, z \leq \zeta x + \eta\}$, for three appropriate real parameters ζ, η, ξ . The query line that contains $\overline{pq}$ can be represented as $L_{a, b, c, d} = \{(x, y, z) | z = ay + b, x = cy + d\}$, for four real parameters a, b, c, d . We can represent (the line bounding) a curtain $C_{\zeta, \eta, \xi}$ as the point (ζ, η, ξ) in a 3-dimensional parametric space Π . A query line $L_{a, b, c, d}$ intersects the negative curtain $C_{\zeta, \eta, \xi}$ if and only if $a\xi + b \leq \zeta(c\xi + d) + \eta$, which is the equation of a halfspace in Π bounded by the hyperbolic paraboloid $\eta = -c\zeta\xi + a\xi - d\zeta + b$.

If we regard η as the third, vertical coordinate axis in Π , then a point (ζ, η, ξ) lies above the paraboloid if and only if the line $L_{a,b,c,d}$ intersects $C^-(e_c^t)$.

We choose some parameter $r \leq n$, and obtain a partition of the (set of points representing the) semi-canonical lines into $O(r)$ subsets $L_1, \dots, L_m$, each set consisting of $O(n/r)$ points, and the dual surface of a query line separates the points of at most $O(r^{2/3+\epsilon})$ sets, for any $\epsilon > 0$. This partitioning follows from the technique developed by Agarwal and Matoušek [5] for range searching where the ranges are semi-algebraic sets.

Given a query paraboloid that represents the query line ℓ , every set L_i either lies entirely under the paraboloid, lies entirely above the paraboloid, or is separated by it. If L_i is below the paraboloid we ignore it. If L_i is above the paraboloid we pass L_i to the next level of the data structure, otherwise we recurse into L_i .

Using this structure we can compute preprocessed sets of triangles that have a negative semi-canonical curtain intersected by $\overline{pq}$. Similarly, we can compute preprocessed sets of triangles that have a positive semi-canonical curtain intersected by $\overline{pq}$.

2.6 Determining an Intersection with Ordinary Curtains

In the last level of our data structure, we are given precomputed sets of triangles, computed in the previous levels. Each such subset T that participates in the output for a query segment $\overline{pq}$, has the property that pq intersects $C^-(e_c^t)$, for every triangle $t \in T$; moreover, the projection of $\overline{pq}$ onto the xy -plane intersects the projection of a second, not necessarily semi-canonical, edge e_r^t of t . It is therefore sufficient to check whether $\overline{pq}$ intersects any of the corresponding positive curtains $C^+(e_r^t)$.

This task is equivalent to testing whether there exists a line in a given set of lines in 3-space (namely, the extensions of the edges e_r^t) which passes below a query line (which is the line containing $\overline{pq}$). Equivalently, it suffices to test whether all the input lines pass above the query line.

This task can be accomplished in $O(n^{1/2}2^{O(\log^+ n)})$ time, with linear space and $O(n \log n)$ preprocessing time, using the shallow cutting lemma of Matoušek [14].

Specifically, let $\mathcal{L}$ be the input set of lines and let ℓ denote the query line. We orient each line in $\mathcal{L}$, as well as the query line ℓ , so that the xy -projection of ℓ lies clockwise to the projections of all lines in $\mathcal{L}$.

The preceding levels of the data structure allow us to assign such orientations to all lines in $\mathcal{L}$, so that the above condition will hold for every query line that gets to interact with $\mathcal{L}$.

We map each line $\lambda \in \mathcal{L}$ to its Plücker hyperplane Π_λ in $\mathbb{R}^5$, and map ℓ to its Plücker point p_ℓ ; see [9,17] for details. Since the xy -projections of ℓ and of the lines of $\mathcal{L}$ are oriented as prescribed above, it follows that ℓ passes below all the lines of $\mathcal{L}$ if and only if p_ℓ lies below all the hyperplanes Π_λ , for $\lambda \in \mathcal{L}$; in other words, p_ℓ has to lie below the lower envelope of these hyperplanes. Since the complexity of this envelope is $O(n^2)$ (being a convex polyhedron in 5-space

with at most n facets), the technique of [14] does indeed yield the performance bounds asserted above.

Similarly we can test whether $\overline{pq}$ intersects some $C^-(e_r^t)$ in a canonical set of triangles where $\overline{pq}$ intersects a positive semi-canonical curtain $C^+(e_c^t)$ of each of its triangles.

The space requirement $\Sigma(n)$ of any level in our data structure (including all the subsequent levels below it), for a set of n triangles, satisfies the recurrence:

$$\Sigma(n) = O(r)\Sigma'(\frac{n}{r}) + O(r)\Sigma(\frac{n}{r}),$$

where $\Sigma'(n)$ is the space requirement of the next levels, for a set of n triangles. If $\Sigma'(n) = O(n^{1+\epsilon})$, for any $\epsilon > 0$, then, choosing r to be a sufficiently large constant that depends on ϵ , one can show that $\Sigma(n) = O(n^{1+\epsilon})$, for any $\epsilon > 0$, as well. This implies that the overall storage required by the data structure is $O(n^{1+\epsilon})$, for any $\epsilon > 0$. The preprocessing time obeys a similar recurrence whose solution is also $O(n^{1+\epsilon})$.

Similarly, the query time $Q(n)$ of any level in our data structure, for a set of n triangles, satisfies the recurrence:

$$Q(n) = O(r) + O(r)Q'(\frac{n}{r}) + O(r^{2/3})Q(\frac{n}{r}),$$

where $Q'(n)$ is the query time at the next levels (for n triangles). If $Q'(n) = O(n^{2/3+\epsilon})$ for any $\epsilon > 0$, then, choosing r as above, it follows that $Q(n) = O(n^{2/3+\epsilon})$, for any $\epsilon > 0$, as well. In conclusion, we thus obtain:

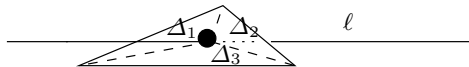
Theorem 1. *The ray shooting problem amid n fat triangles in $\mathbb{R}^3$ can be solved with a query time $O(n^{2/3+\epsilon})$, using a data structure of size $O(n^{1+\epsilon})$, which can be constructed in $O(n^{1+\epsilon})$ time, for any $\epsilon > 0$.*

Combining this result which uses linear space with a result for answering ray shooting queries for general triangles in logarithmic time using $O(n^{4+\epsilon})$ space (there are no better results for the special cases that we discuss), a trade-off between storage and query time can be obtained, as in [6]. Thus, we have:

Theorem 2. *For any parameter $n \leq m \leq n^4$, the ray shooting problem for a set of n fat triangles, or n triangles stabbed by a common line, can be solved using $O(m^{1+\epsilon})$ space and preprocessing we can answer a query in $O(n^{8/9+\epsilon}/m^{2/9})$ query time, for any $\epsilon > 0$.*

2.7 Ray Shooting amid Triangles Stabbed by a Common Line

We now adapt our algorithm to perform ray shooting amid triangles that are stabbed by one common line. Observe that a triangle intersecting a line ℓ can be covered by three triangles, each having two edges with an endpoint on ℓ , as illustrated in the figure:



Our algorithm can be easily adapted to perform ray shooting amid triangles, each having two edges that end on a fixed given line.

In fact, the only modification needed to be done is in the representation of the semi-canonical edges, and in finding intersection with semi-canonical curtains. Assume that the triangles are stabbed by the z -axis. This assumption can be met by a proper rigid transform. In this case, the representation of a semi-canonical curtain becomes: $C_{\zeta,\eta,\xi} = \{(x, y, z) | z = \zeta x + \eta, y = \xi x\}$. Again, this curtain can be represented as the point (ζ, η, ξ) in a 3-dimensional parametric space Π .

A query line $L_{a,b,c,d} = \{(x, y, z) | z = ax + b, y = cx + d\}$ intersects a negative curtain $C_{\zeta,\eta,\xi}$ if and only if $(\eta\xi + d\zeta + bc - da - b\xi - \eta c)(\xi - c) \geq 0$. The first factor is the equation of a half space bounded by a hyperbolic paraboloid in Π .

We partition the points in Π in the same manner described earlier for the case of fat triangles. The ranges with respect to which the partitioning is constructed are somewhat more complicated, but the technique of [5] applies in this case too. The rest of the algorithm remains the same.

In fact, this algorithm can be adapted to any set of triangles, each having two edges that can each be described with three parameters (or where each triangle can be covered by such triangles). For example, triangles stabbed by an algebraic curve of small degree, triangles tangent to some sphere, etc. In all these cases we obtain an algorithm that requires $O(n^{1+\epsilon})$ storage and preprocessing, and answers ray shooting queries in $O(n^{2/3+\epsilon})$ time.

3 Stone Throwing amid Arbitrary Triangles

If any one of you is without sin, let him be the first to throw a stone ... (John 8:7).

Next we study another problem of shooting along arcs amid triangles in three dimensions, which we refer to as *stone throwing*. In this problem we are given a set $\mathcal{T}$ of n triangles in $\mathbb{R}^3$, and we wish to preprocess them into a data structure that can answer efficiently stone throwing queries, where each query specifies a point $p \in \mathbb{R}^3$ and a velocity vector $\mathbf{v} \in \mathbb{R}^3$; these parameters define a parabolic trajectory traced by a stone thrown from p with initial velocity $\mathbf{v}$ under gravity (which we assume to be exerted in the negative z -direction), and the query asks for the first triangle of $\mathcal{T}$ to be hit by this trajectory.

As noted in the introduction, the parabola π that contains the stone trajectory has five degrees of freedom and can be represented by the quintuple (a, b, c, d, e) that define the equations $y = ax + b$, $z = cx^2 + dx + e$ of Π . The first equation defines the vertical plane V_Π that contains π . Note that, under gravity, we have $c < 0$, i.e., π is *concave*.

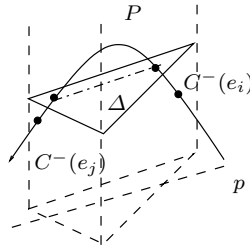
Unlike the case of ray shooting, we only consider here the basic case where the triangles of $\mathcal{T}$ are arbitrary, and present a solution that uses near-linear storage and preprocessing, and answers stone-throwing queries in time near $n^{3/4}$.

Using the parametric searching technique, as in [4], we can reduce the problem to testing emptiness of concave vertical parabolic arcs, in which we wish to determine whether such a query arc intersects any triangle in $\mathcal{T}$.

Lemma 3. *Given a non-vertical triangle Δ , contained in a plane h , with three edges e_1 , e_2 and e_3 , and given a parabolic arc $\tilde{p}q$ contained in some concave vertical parabola π , and delimited by the points p, q , all in $\mathbb{R}^3$, then the arc $\tilde{p}q$ intersects the triangle Δ if and only if (exactly) one of the following conditions holds:*

- (i) $\tilde{p}q$ intersects one positive curtain $C^+(e_i)$ and one negative curtain $C^-(e_j)$ of Δ .
- (ii) One endpoint, say p , is below Δ and $\tilde{p}q$ intersects one positive curtain $C^+(e_i)$ of Δ .
- (iii) One endpoint, say p , is above Δ and $\tilde{p}q$ intersects one negative curtain $C^-(e_i)$ of Δ .
- (iv) One endpoint, say p , is above Δ and q is below Δ , or vice versa.
- (v) The parabola π intersects the plane h , $\tilde{p}q$ intersects two negative curtains $C^-(e_i)$ and $C^-(e_j)$, at the respective intersection points p_1 and p_2 and $S(p_1) \leq \text{slope}(h \cap V_\pi) \leq S(p_2)$ (or vice versa), where $S(x)$ is the slope of the tangent to π at point x , and $\text{slope}(h \cap V_\pi)$ is the slope of this line within the vertical plane V_π .
- (vi) One endpoint, say p lies below Δ , π intersects the plane h , $\tilde{p}q$ intersects one negative curtain $C^-(e_i)$ of Δ at some point p_1 , and $S(p_1) \leq \text{slope}(h \cap V_\pi) \leq S(p)$, or $S(p) \leq \text{slope}(h \cap V_\pi) \leq S(p_1)$.
- (vii) The parabola π intersects the plane h , p and q are below Δ and $S(p) \leq \text{slope}(h \cap V_\pi) \leq S(q)$, (or vice versa).

Proof. Straightforward; the first four conditions are similar to the ones given in Lemma 1. The fifth condition is depicted in below, and the last two conditions are similar to it.



As in the case of ray shooting, we can use Lemma 3 to devise an algorithm that solves the parabolic arc emptiness problem by testing whether any of the conditions (i)–(vii) holds. For lack of space, we sketch briefly how to check for condition (v). The other conditions can be handled in a similar manner.

This test is somewhat more involved than the preceding one. It also constructs a multi-level data structure, whose top levels are similar to the entire structure of the preceding data structure. Using them, we can collect, for a query arc γ , canonical subsets of triangles, so that, for each such set $\mathcal{T}'$, γ intersects the negative curtains erected over both edges e_1^t, e_2^t of every triangle $t \in \mathcal{T}'$. It remains to test, for each such output set $\mathcal{T}'$, whether there exists $t \in \mathcal{T}'$, such

that the parabola π containing γ intersects the plane containing t , and the two slope conditions, over $C^-(e_1^t)$ and $C^-(e_2^t)$ are satisfied.

The next level of the structure collects all triangles whose supporting planes are crossed by π . Each such plane has three degrees of freedom, and can be represented as a point in dual 3-space. Each concave vertical parabola π is mapped to a surface in that space, representing all planes that are tangent to π . Again, arguing as above, we can apply the partitioning technique of [5] to construct a partition tree over $\mathcal{T}'$, using which we can find all triangles whose planes are crossed by π as a collection of canonical subsets.

The next levels test for the slope conditions. Consider the slope condition over $C^-(e_1^t)$, for a triangle t in a canonical subset $\mathcal{T}''$ (that is, the condition $S(p_1) \leq \text{slope}(h \cap V_\pi)$). There are two slopes that need to be compared. The first is the slope $S(p_1)$ of the tangent to π at the point p_1 where it crosses $C^-(e_1^t)$. This slope depends only on two of the parameters that specify t , namely the coefficients of the equation of the xy -projection of e_1^t . The second slope is that of the line $h \cap V_\pi$, which depends only on the equation of the plane h containing t . Moreover, if the equation of this plane is $z = \xi x + \eta y + \zeta$, then the slope is independent of ζ . In other words, the overall slope condition can be expressed as a semi-algebraic condition that depends on only *four* parameters that specify t . Hence, we can represent the triangles of $\mathcal{T}''$ as points in an appropriate 4-dimensional parametric space, and map each parabola π into a semi-algebraic set of so-called *constant description complexity* [16] in that space, which represents all triangles t for which π satisfies the slope condition over $C^-(e_1^t)$. We now apply the partitioning technique of [5] for the set of points representing the triangles of $\mathcal{T}''$ and for the set of ranges corresponding to parabolas π as just defined. It partitions $\mathcal{T}^*$ into r subsets, each consisting of $O(n/r)$ points, so that any surface σ_π separates the points in at most $O(r^{3/4+\epsilon})$ subsets, for any $\epsilon > 0$. This result depends on the existence of a vertical decomposition of the four-dimensional arrangement of the m surfaces σ_π into $O(m^{4+\epsilon})$ elementary cells (see [5] for details), which follows from a recent result of Koltun [13]. More details are given in the full version. The slope condition over the other negative curtains $C^-(e_2^t)$ is handled in the next and final level of the data structure, in exactly the same way as just described. We omit the further technical but routine details of handling these levels.

Since each level of the data structure deals with sets of points in some parametric space of dimension at most four, the preceding analysis implies that the overall query time is $O(n^{3/4+\epsilon})$, for any $\epsilon > 0$, and the storage remains $O(n^{1+\epsilon})$, for any $\epsilon > 0$.

Omitting all further details, we thus obtain:

Theorem 3. *A set of n triangles in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O(n^{1+\epsilon})$ in time $O(n^{1+\epsilon})$, for any $\epsilon > 0$, so that any stone-throwing query can be answered in time $O(n^{3/4+\epsilon})$.*

Remark: As mentioned in the introduction, this result can be extended to shooting along arcs that are graphs of univariate algebraic functions of constant maximum degree that lie in any vertical plane. We simply break such a graph into its

maximal convex and concave portions and apply a machinery similar to the one described above to each portion separately. We note that the method prescribed above can be easily extended to handle shooting along any concave vertical arc (of bounded algebraic degree). The case of convex arcs is handled by reversing the direction of the z -axis (based on a similar flipped version of Lemma 3).

References

1. P. K. Agarwal. Applications of a new space partition technique. In *Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes Comput. Sci.*, pages 379–392, 1991.
2. P. K. Agarwal. *Intersection and Decomposition Algorithms for Planar Arrangements*. Cambridge University Press, New York, USA, 1991.
3. P. K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. *SIAM J. Comput.*, 21:540–570, 1992.
4. P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.
5. P. K. Agarwal and J. Matoušek. On range searching with semi-algebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994.
6. P. K. Agarwal and M. Sharir. Applications of a new space-partitioning technique. *Discrete Comput. Geom.*, 9:11–38, 1993.
7. P. K. Agarwal and M. Sharir. Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions. *SIAM J. Comput.*, 25:100–116, 1996.
8. B. Chazelle, H. Edelsbrunner, M. Grigni, L. J. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12:54–68, 1994.
9. B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi. Lines in space: Combinatorics and algorithms. *Algorithmica*, 15:428–447, 1996.
10. B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. *Discrete Comput. Geom.*, 4:551–581, 1989.
11. M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica*, 12:30–53, 1994.
12. D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, volume 443 of *Lecture Notes Comput. Sci.*, pages 400–413. Springer-Verlag, 1990.
13. V. Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 56–65, 2001.
14. J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2(3):169–186, 1992.
15. S. Mohabian and M. Sharir. Ray shooting amidst spheres in three dimensions and related problems. *SIAM J. Comput.*, 26:654–674, 1997.
16. M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, New York, 1995.
17. J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations*. Academic Press, New York, NY, 1991.

Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs*

Aleksandrs Slivkins

Cornell University, Ithaca NY 14853, USA
slivkins@cs.cornell.edu

Abstract. Given a graph and pairs $s_i t_i$ of terminals, the edge-disjoint paths problem is to determine whether there exist $s_i t_i$ paths that do not share any edges. We consider this problem on acyclic digraphs. It is known to be NP-complete and solvable in time $n^{O(k)}$ where k is the number of paths. It has been a long-standing open question whether it is fixed-parameter tractable in k . We resolve this question in the negative: we show that the problem is $W[1]$ -hard. In fact it remains $W[1]$ -hard even if the demand graph consists of two sets of parallel edges. On a positive side, we give an $O(m + k! n)$ algorithm for the special case when G is acyclic and $G + H$ is Eulerian, where H is the demand graph. We generalize this result (1) to the case when $G + H$ is “nearly” Eulerian, (2) to an analogous special case of the unsplittable flow problem. Finally, we consider a related NP-complete routing problem when only the first edge of each path cannot be shared, and prove that it is fixed-parameter tractable on directed graphs.

1 Introduction

Given a graph G and k pairs $(s_1, t_1), \dots, (s_k, t_k)$ of terminals, the edge-disjoint paths problem is to determine whether there exist $s_i t_i$ paths that do not share any edges. It is one of Karp’s original NP-complete problems [8]. Disjoint paths problems have a great theoretical and practical importance; see [7, 11, 19] for a comprehensive survey.

The problem for a bounded number of terminals have been of particular interest. For undirected graphs, Shiloach [16] gave an efficient polynomial-time algorithm for $k = 2$, and Robertson and Seymour [14] proved that the general problem is solvable in time $O(f(k)n^3)$. The directed edge-disjoint paths problem was shown NP-hard even for $k = 2$ by Fortune, Hopcroft and Wyllie [6]. On acyclic digraphs the problem is known to be NP-complete and solvable in time $O(knm^k)$ [6].

Since 1980 it has been an interesting open question whether a better algorithm is possible for acyclic graphs. We should not hope for a polynomial-time algorithm, but can we get rid of k in the exponent and get a running time of $O(f(k)n^c)$ for some constant c , as Robertson and Seymour do for the undirected case? Such algorithms are called *fixed-parameter tractable*.

We resolve this open question in the negative using the theory of fixed-parameter tractability due to Downey and Fellows [4]. Specifically, we show that the directed edge-disjoint paths problem on acyclic graphs is $W[1]$ -hard in k .

* This work has been supported in part by a David and Lucile Packard Foundation Fellowship and NSF ITR/IM Grant IIS-0081334 of Jon Kleinberg.

Fixed-parameter tractability. A problem is *parameterized* by $k \in \mathbb{N}$ if its input is a pair (x, k) . Many NP-hard problems can be parameterized in a natural way; e.g. the edge-disjoint paths problem can be parameterized by the number of paths. Efficient solutions for small values of the parameter might be useful. Call a decision problem P *fixed-parameter tractable* in k if there is an algorithm that for every input (x, k) decides whether $(x, k) \in P$ and runs in time $O(|x|^c f(k))$ for some constant c and some computable function f .

Proving that some NP-complete parameterized problem is *not* fixed-parameter tractable would imply that $P \neq NP$. However, Downey and Fellows [4] developed a technique for showing *relativized* fixed-parameter intractability. They use reductions similar to those for NP-completeness. Suppose there is a constant c and computable functions f, g such that there is a reduction that maps every instance (x, k) of problem P to an instance $(y, f(k))$ of problem Q , running in time $O(g(k)|x|^c)$ and mapping “yes” instances to “yes” instances and “no” instances to “no” instances (we call it a *fixed-parameter reduction*). Then if P is fixed-parameter intractable then so is Q .

There are strong reasons to believe that the problem k -CLIQUE of deciding for a given undirected graph G and an integer k whether G contains a clique of size k is not fixed-parameter tractable [4]. Recently Downey et al. [3] gave a simpler (but weaker) alternative justification based on the assumption that there is no algorithm with running time $2^{o(n)}$ that determines, for a Boolean circuit of total description size n , whether there is a satisfying input vector.

Existence of a fixed-parameter reduction from k -CLIQUE to some problem P is considered to be an evidence of fixed-parameter intractability of P . Problems for which such reduction exists are called $W[1]$ -hard, for reasons beyond the scope of this paper. For a thorough treatment of fixed-parameter tractability see Downey and Fellows [4].

Our contributions: disjoint paths. All routing problems in this paper are parameterized by the number k of terminal pairs. Given a digraph $G = (V, E)$ and terminal pairs $\{s_i, t_i\}$ the *demand graph* H is a digraph on a vertex set V with k edges $\{t_i s_i\}$. Note that H can contain parallel edges. Letting d^{in} and d^{out} be the in- and out-degree respectively, a digraph is called *Eulerian* if for each vertex $d^{in} = d^{out}$. The *imbalance* of a digraph is $\frac{1}{2} \sum_v |d^{out}(v) - d^{in}(v)|$.

Above we claimed that the directed edge-disjoint paths problem on acyclic graphs is $W[1]$ -hard. In fact, we show that it is so even if H consists of two sets of parallel edges; this case was known to be NP-complete [5,18]. Our proof carries over to the node-disjoint version of the problem.

On the positive side, recall that for a general H the problem is solvable in time $n^{O(k)}$ by [6]. We show a special case which is still NP-complete but fixed-parameter tractable. Specifically, consider the directed edge-disjoint paths problem if G is acyclic and $G + H$ is Eulerian. This problem is NP-complete (Vygen [18]). We give an algorithm with a running time $O(m + k! n)$.¹ This extends to the running time of $O(m + (k + b)! n)$ on general acyclic digraphs, where b is the imbalance of $G + H$.

¹ This problem is equivalent to its undirected version [18], hence is fixed-parameter tractable due to Robertson and Seymour [14]. However, as they observe in [7], their algorithm is extremely complicated and completely impractical even for $k = 3$.

Our contributions: unsplittable flows. We consider the unsplittable flow problem [9], a generalized version of disjoint paths that has capacities and demands. The instance is a triple (G, H, w) where w is a function from $E(G \cup H)$ to positive reals and $w(t_i s_i)$ is the demand on the i -th terminal pair. The question is whether there are $s_i t_i$ paths such that for each edge e of G the capacity w_e is greater or equal to the sum of demands of all paths that come through e . The edge-disjoint paths problem is a special case of the unsplittable flow problem with $w \equiv 1$.

The unsplittable flow problem can model a variety of problems in virtual-circuit routing, scheduling and load balancing [9,12]. There has been a number of results on approximation [9,12,2,17,1]. Most relevant to this paper is the result of Kleinberg [10] that the problem is fixed-parameter tractable on undirected graphs if all capacities are 1 and all demands are at most $\frac{1}{2}$.

We show that the unsplittable flow problem is $W[1]$ -hard on acyclic digraphs even if H is a set of parallel edges. If furthermore all capacities are 1 the problem is still NP-hard [9] (since for a two-node input graph with multiple edges it is essentially a bin-packing problem). We show it is fixed-parameter tractable with a running time of $O(e^k)$ plus one max-flow computation. However, the problem becomes $W[1]$ -hard again if there are (a) *three* sink nodes, even if all demands are at most $\frac{1}{2}$, (b) two source nodes and two sink nodes, even if all demands are *exactly* $\frac{1}{2}$. This should be contrasted with the result of [10].

Moreover we show that similarly to disjoint paths, the unsplittable flow problem (a) can be solved in time $O(knm^k)$ if G is directed acyclic, (b) becomes fixed-parameter tractable if furthermore $G + H$ is *Eulerian under w* , that is if for each node the total weight of incoming edges is equal to the total weight of outgoing edges. The running time for the latter case is $O(m + k^{4k}n)$.

Our contributions: first-edge-disjoint paths. We looked for $s_i t_i$ paths under the constraint of edge-disjointness. To probe the boundary of intractability, we now relax this constraint and show that the associated routing problem is (still) NP-complete on acyclic digraphs but fixed-parameter tractable (even) on general digraphs. This problem turns out to capture a model of starvation in computer networks, see Section 4.2 of the full version of this paper.

Call two paths *first-edge-disjoint* if the first edge of each path is not shared with the other path. The first-edge-disjoint paths problem is to determine whether there exist $s_i t_i$ paths that are first-edge-disjoint. We show that on digraphs any instance of this problem can be reduced, in polynomial time, to an instance whose size depends only on k . With some care we get the running time $O(mk + k^5(ek)^k)$.

Further directions. Given our hardness result for edge-disjoint paths, we hope to address the case when G is acyclic and planar. This problem is still NP-complete [18], but becomes polynomial if furthermore $G + H$ is planar (this follows from [13], e.g. see [19]). The directed *node*-disjoint paths problem on planar graphs is solvable in polynomial time for every fixed k due to A. Schrijver [15].

Notation. Terminals s_i, t_i are called sources and sinks, respectively. Each terminal is located at some node; we allow multiple terminals to be located at the same node. We call a node at which one or more sources is located a *source node*. Similarly a node with one or more sinks is a *sink node*. Note that a node can be both a source and a sink node.

Both source and sink nodes are called *terminal nodes*. If no confusion arises, we may use a terminal name to refer to the node at which the terminal is located.

We parameterize all routing problems by the number k of terminal pairs. We denote the number of nodes and edges in the input graph G by n and m respectively.

Organization of the paper. In Section 2 we present our hardness results, Section 3 is on the algorithmic results, and Section 4 is on the first-edge-disjoint paths problem. Due to the space constraints, some proofs are deferred to the full version of this paper available at <http://www.cs.cornell.edu/people/slivkins/research/>.

2 Hardness Results

Theorem 1. *The edge-disjoint paths problem is $W[1]$ -hard on acyclic digraphs.*

Proof: We define a fixed-parameter reduction from k -CLIQUE to the directed edge-disjoint paths problem on acyclic digraphs. Let (G, k) be the instance of k -CLIQUE, where $G = (V, E)$ is an undirected graph without loops. We construct an equivalent instance (G', k') of the directed edge-disjoint paths problem where G' is a directed acyclic graph and $k' = k(k+1)/2$. Denote $[n] = \{1 \dots n\}$ and assume $V = [n]$.

Idea. We create a $k \times n$ array of identical gadgets. Intuitively we think of each row as a copy of V . For each row there is a path ('selector') that goes through all gadgets, skipping at most one and killing the rest, in the sense that other paths cannot route through them. This corresponds to selecting one gadget (and hence one vertex of G) from each row. The selected vertices form a multi-set of size k . We make sure that we can select a given multi-set if and only if it is a k -clique in G . Specifically, for each pair of rows there is a path ('verifier') that checks that the vertices selected in these rows are connected in G . Note that this way we don't need to check separately that the selected vertices are distinct.

Construction. We'll use k paths P_i ('selectors') and $\binom{k}{2}$ paths $P_{ij}, i < j$ ('verifiers'). Denote the terminal pairs by $s_i t_i$ and $s_{ij} t_{ij}$ respectively. Selector P_i will select one gadget from row i ; verifier P_{ij} will verify that there is an edge between the vertices selected in rows i and j .

Denote gadgets by $G_{iu}, i \in [k], u \in V$. The terminals are distinct vertices not contained in any of the gadgets. There are no edges from sinks or to sources. We draw the array of gadgets so that row numbers increase downward, and column numbers increase to the right. Edges between rows go *down*; within the same row edges go *right*.

We start with the part of construction used by verifiers. Each gadget G_{iu} consists of $k-1$ parallel paths $(a_r, b_r), r \in [k] - \{i\}$. For each s_{ij} there are edges $s_{ij} a_j$ to every gadget in row i . For each t_{ij} there are edges $b_i t_{ij}$ from every gadget in row j (Fig. 1a); there will be no more edges from s_{ij} 's or to t_{ij} 's. To express the topology of G , for each edge uv in G and each $i < j$ we create an edge from b_j in G_{iu} to a_i in G_{jv} (Fig. 1b). There will be no more edges or paths between rows. The following lemma explains what we have done so far.

Lemma 1. (a) *Suppose we erase all gadgets in rows i and j except G_{iu} and G_{jv} . Then an $s_{ij} t_{ij}$ path exists if and only if $uv \in E$.* (b) *Suppose we select one gadget in each row and erase all others. Then there exist edge-disjoint paths from s_{ij} to t_{ij} for each $i, j \in [k], i < j$, if and only if the selected gadgets correspond to a k -clique in G .*

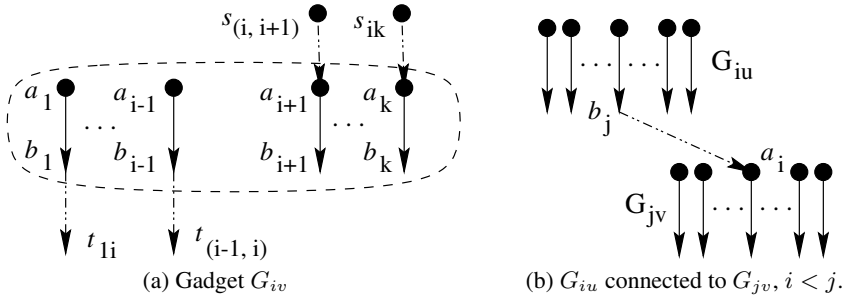


Fig. 1. Gadgets and verifiers

We could prove Lemma 1 right away, but we will finish the construction first. Recall that each gadget consists of $k - 1$ parallel wires (a_r, b_r) . Each wire is a simple path of length 3: (a_r, a'_r, b'_r, b_r) (Fig. 2a). Let “level 1” be the set of all a_r and a'_r (in all wires and in all gadgets). Let “level 2” be the set of all b'_r and b_r . Each selector enters its row at level 1. The idea is that the only way it can skip a gadget is by going from level 1 to level 2, so, since within a given row there is no path back to level 1, at most one gadget can be skipped. The remainder of the construction makes this concrete.

First we complete the construction of a single gadget (Fig. 2a). In each gadget G_{iu} there are two edges from each wire r to the next one, one for each level. For $r \neq i - 1, i$ these are $(a'_r a_{r+1})$ and $(b_r b'_{r+1})$ (note that there is no wire i). The edges between wires $i - 1$ and $i + 1$ are $(a'_{i-1} a_{i+1})$ and $(b_{i-1} b'_{i+1})$.

It remains to connect gadgets within a given row i (Fig. 2b). There are edges from s_i to a_1 in G_{i1} , and from b_k in G_{in} to t_i . There are two edges from each gadget to the next one, one for each level: from a'_k to a_1 and from b_k to b'_1 . Finally, there are *jumps* over any given gadget in the row: an edge from s_i to b'_1 of G_{i2} jumps over G_{i1} , edges from a'_k of $G_{(i,u-1)}$ to b'_1 of $G_{(i,u+1)}$ jump over G_{iu} , and an edge from a'_k in $G_{(i,n-1)}$ to t_i jumps over G_{in} .

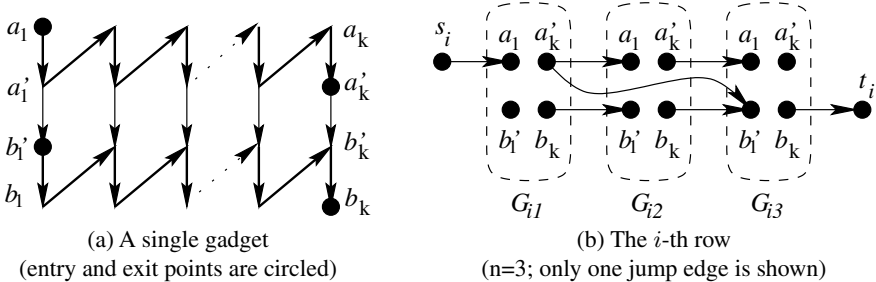


Fig. 2. Additional wiring for selectors

Proof of correctness. First we check that our construction is acyclic. It suffices to provide a topological ordering. For $i \in [k]$ and $j \in [2]$, let Q_{ij} be the ordering of vertices

in the level j path in row i , i.e. Q_{i1} is the unique path from a_1 in G_{i1} to a'_k in G_{in} and Q_{i2} is the unique path from b'_1 in G_{i1} to b_k in G_{in} . Then the required ordering is given by (all sources; $Q_{11}, Q_{12}; Q_{21}, Q_{22}; \dots; Q_{k1}, Q_{k2}$; all terminals).

Now we prove Lemma 1. We stated part (a) for intuition only. The proof is obvious. For part (b), the 'if' direction is now straightforward since each gadget assigns a separate wire to each verifier than can potentially route through it, and the wires corresponding to a given verifier are connected in the right way. For the 'only if' direction, note that there is at most one edge between any given pair of gadgets in different rows, so the total number of edges between the selected gadgets is at most $\binom{k}{2}$. In fact it is exactly $\binom{k}{2}$ since each verifier has to use at least one of these edges. Therefore any pair of selected gadgets is connected, which happens if and only if the corresponding vertices are connected in G . Claim proved.

Lemma 2. *For each possible $s_i t_i$ path there is a gadget such that verifiers cannot enter all other gadgets in row i .*

Proof: All edges between rows go "down", so if P_i ever leaves row i , it can never come back up. Thus P_i must stay in row i and visit each gadget in it successively, possibly jumping over one of them. If P_i enters a given gadget at a_1 , it can either route through level 1 and exit at a'_k , or switch to level 2 somewhere in the middle and exit at b_k . If P_i enters at b'_1 , it must route through level 2 and exit at b_k .

P_i starts out at level 1. If it never leaves level 1 then it uses up every edge $a_r a'_r$ (so verifiers cannot enter any gadget in the row). Else it switches to level 2, either within a gadget or by jumping over a gadget, call it G_{iu} . To the left of G_{iu} all edges $a_r a'_r$ are used by P_i , so verifiers cannot enter. To the right of G_{iu} the selector uses all edges $b'_r b_r$, so verifiers cannot exit the row from any G_{iv} , $v > u$. If a verifier enters such gadget it never leaves the row since within a row inter-gadget edges only go right. Therefore verifiers cannot enter gadgets to the right of G_{iu} , either. $\square$

We need to prove that our construction is a positive instance of the directed edge-disjoint paths problem if and only if (G, k) is a positive instance of k -CLIQUE. For the "if" direction, let $u_i \dots u_k$ be a k -clique in G , let each selector P_i jump over G_{iu_i} and apply Lemma 1b. For the "only if" direction, suppose our construction has a solution. By Lemma 2 verifiers use only one gadget in each row (that is, all verifiers use the same gadget). Therefore by Lemma 1b these gadgets correspond to a k -clique in G . This completes the proof of Thm. 1. $\square$

Now we extend our result by restricting the demand graph.

Theorem 2. *On acyclic digraphs, (a) the edge-disjoint paths problem is $W[1]$ -hard even if the demand graph consists of two sets of parallel edges, (b) the unsplittable flow problem is $W[1]$ -hard even if the demand graph is a set of parallel edges.*

Proof: (Sketch) In the construction from the proof of Thm. 1, contract all s_i, s_{ij}, t_i and t_{ij} to s, s', t and t' , respectively. Clearly each selector has to start in a distinct row; let P_i be the selector that starts in row i . Since there is only one edge to t from the k -th row, P_{k-1} has to stay in row $k-1$. Iterating this argument we see that each P_i has to stay in row i , as in the original construction. So Lemma 2 carries over. Each $s' t'$ path has

to route between some pair of rows, and there are at most $\binom{k}{2}$ edges between selected gadgets. This proves Lemma 1b and completes part (a).

For part (b) all edges incident to s' or t' and all edges between rows are of capacity 1; all other edges are of capacity 2. Each verifier has demand 1, each selector has demand 2. Contract s' to s and t' to t . $\square$

Kleinberg [10] showed that the *undirected* unsplittable flow problem apparently becomes more tractable when the maximal demand is at most a half of the minimal capacity. The next theorem shows that on acyclic digraphs this does not seem to be the case; the proof is deferred to the full version of this paper.

Theorem 3. *On acyclic digraphs, if all capacities are 1 and all demands are at most $\frac{1}{2}$, the unsplittable flow problem is $W[1]$ -hard even if there are only (a) two source nodes and two sink nodes, (b) one source node and three sink nodes. Moreover, the first result holds even if all demands are exactly $\frac{1}{2}$.*

3 Algorithmic Results

In this section G is a directed graph on n vertices, and H is the demand graph with k edges. Let S_k be the group of permutations on $[k] = \{1 \dots k\}$. Assuming a fixed numbering $s_1 t_1 \dots s_k t_k$ of terminal pairs, if for some permutation $\pi \in S_k$ there are edge-disjoint $s_i t_{\pi(i)}$ paths, $i \in [k]$, then we say that these paths are *feasible* and *realize* π . Let $\Pi(G, H)$ be the set of all such permutations. By abuse of notation we consider it to be a $k!$ -bit vector.

Theorem 4. *Suppose G is acyclic and $G + H$ is Eulerian. Then we can compute $\Pi = \Pi(G, H)$ in time $O(m + k!n)$. In particular, this solves the directed edge-disjoint paths problem on (G, H) .*

Proof: In G , let u be the vertex of zero in-degree, and let $v_1 \dots v_r$ be the vertices adjacent to u . Since $G + H$ is Eulerian, there are exactly r sources sitting in u , say $s_{i_1} \dots s_{i_r}$. Therefore each feasible path allocation induces k edge-disjoint paths on $G' = G - u$ such that there is a (unique) path that starts at each v_i . This observation suggests to consider a smaller problem instance (G', H') where we obtain the new demand graph H' from H by moving each s_{i_j} from u to v_j . The idea is to compute Π from $\Pi' = \Pi(G', H')$ by gluing the uv_i paths with paths in (G', H') .

Formally, let H' be the demand graph associated with terminal pairs $s'_1 t_1 \dots s'_k t_k$ where $s'_i = v_j$ if $i = i_j$ for some j , and $s'_i = s_i$ otherwise. Then, obviously, G' is acyclic and $G' + H'$ is Eulerian. Let $I = \{i_1 \dots i_r\}$ and $S_I \subset S_k$ be the subgroup of all permutations on I extended to identity on $[k] - I$. We claim that

$$\Pi = \{\pi' \circ \sigma : \pi' \in \Pi' \text{ and } \sigma \in S_I\} \quad (1)$$

Indeed, let $\sigma \in S_I$ and $\pi' \in \Pi'$; let $P'_1 \dots P'_k$ be paths that realize π' in (G', H') . Then $P_i = s_i s'_{\sigma(i)} \cup P'_{\sigma(i)}$ is a path from s_i to $t_{\pi(\sigma(i))}$ for all i (note that $P_i = P'_i$ for $i \notin I$). Paths $P_1 \dots P_k$ are edge-disjoint, so $\pi' \circ \sigma \in \Pi$.

Conversely, let $\pi \in \Pi$ and $P_1 \dots P_k$ be paths that realize it. The same paths restricted to G' realize some $\pi' \in \Pi'$. For each $i \in I$ the path P_i goes through some s'_j , say through $s'_{\sigma(i)}$. Let $\sigma(i) = i$ for $i \notin I$. Then $\sigma \in S_I$ and $\pi = \pi' \circ \sigma$. Claim proved.

We compute Π by iterating (1) n times on smaller and smaller graphs. To choose u we maintain the set of vertices of zero in-degree; recomputing it after each iteration takes time $O(k)$. To compute (1) in time $O(k!)$ we project Π' to $[k] - I$. $\square$

Recall that we defined the *imbalance* of a graph as $\frac{1}{2} \sum_v |d^{out}(v) - d^{in}(v)|$. Suppose $G + H$ is 'nearly' Eulerian in the sense that its imbalance b is small. We can add b new terminal pairs st where s, t are new vertices, along with a few edges from s to G and from G to t , to get a new problem instance (G', H') such that G' is acyclic and $G' + H'$ is Eulerian. It is easy to see that (G, H) and (G', H') are in fact equivalent (Thm. 2 in [18]). This proves:

Theorem 5. *The edge-disjoint paths problem on acyclic digraphs can be solved in time $O((k+b)! n + m)$, where b is the imbalance of $G + H$.*

Now we will extend the argument of Thm. 4 to the unsplittable flow problem. Recall that an instance of the unsplittable flow problem is a triple (G, H, w) where w is a function from $E(G \cup H)$ to positive reals. Let $d_i = w(t_i s_i)$ be the demand on the i -th terminal pair.

We will need a more complicated version of $\Pi(G, H)$. Let σ and τ be onto functions from $[k]$ to source and sink nodes respectively. Say (σ, τ) is a *feasible pair* if $\sum_{\sigma_i=s} d_i = \sum_{s_i=s} d_i$ for each source node s and $\sum_{\tau_i=t} d_i = \sum_{t_i=t} d_i$ for each sink node t . In other words, a feasible pair rearranges s_i 's on the source nodes and t_i 's on the sink nodes without changing the total demand on each source or sink node. Say paths $P_1 \dots P_k$ realize a feasible pair (σ, τ) if these paths form a solution to the unsplittable flow problem on G with terminal pairs $\sigma_i \tau_i$ and demands w_i . Let $\Pi(G, H, w)$ be the set of all such feasible pairs.

Theorem 6. *Let (G, H, w) be an instance of the unsplittable flow problem such that G is acyclic and $G + H$ is Eulerian under w . Then we can compute $\Pi = \Pi(G, H, w)$ in time $O(m + k^{4k} n)$. In particular, this solves the unsplittable flow problem on (G, H, w) .*

Proof: (Sketch) The proof is similar to that of Thm. 4. Again, letting u be a vertex of zero in-degree in G , the idea is to compute Π from a problem instance on a smaller graph $G' = G - u$.

We derive a problem instance (G', H', w') on k terminal pairs $s'_i t_i$ with demands d_i and capacities given by w . Again, letting $v_1 \dots v_r$ be the nodes adjacent to u in G , the new demand graph H' is obtained from H by moving all sources from u to v_i 's, arranging them in any (fixed) way such that $G' + H'$ is Eulerian under w' . It is easy to see that we get such arrangement from any set of paths that realizes some feasible pair. If such arrangement exists we can find it using enumeration; else Π is empty.

Similarly to (1), we compute Π from $\Pi(G', H', w')$ by gluing the uv_i paths with paths in G' , except now we only consider uv_i paths that respect the capacity constraints on edges uv_i . $\square$

Returning to the general acyclic digraphs, we extend the $n^{O(k)}$ algorithm of [6] from disjoint paths to unsplittable flows.

Theorem 7. *The unsplittable flow problem on acyclic digraphs can be solved in time $O(knm^k)$.*

Proof: We extend the pebbling game from [6]. For each $i \in [k]$ add nodes s'_i and t'_i and infinite-capacity edges $s'_i s_i$ and $t_i t'_i$. Define the pebbling game as follows. Pebbles $p_1 \dots p_k$ can be placed on edges. Each pebble p_i has weight d_i . The *capacity constraint* is that at any moment the total weight of all pebbles on a given edge e is at most w_e . If a pebble p_i sits on edge e , define the *level* of p_i to be the maximal length of a path that starts with e . Pebble p_i can move from edge uv to edge vw if and only if p_i has the highest level among all pebbles and the capacity constraint on vw will be satisfied after the move. Initially each p_i is on $s'_i s_i$. The game is won if and only if each p_i is on $t_i t'_i$.

It is easy to see that the pebbling game has a winning strategy if and only if there is a solution to the unsplittable flow problem (paths in the unsplittable flow problem correspond to trajectories of pebbles). The crucial observation is that if some pebbles visit an edge e then at some moment all these pebbles are on e .

Let G_{state} be the state graph of the pebbling game, with nodes corresponding to possible configurations and edges corresponding to legal moves. The algorithm is to search G_{state} to determine whether the winning configuration is reachable from the starting one. The running time follows since there are m^k possible configurations and at most kn legal moves from each. $\square$

Finally, we consider the case when the demand graph is just a set of parallel edges.

Lemma 3. *If the demand graph is a set of parallel edges and all capacities are 1, the unsplittable flow problem on directed or undirected graphs can be solved in time $O(e^k)$ plus one max-flow computation.²*

Proof: Let s, t be the source and the sink node respectively. Consider some minimal st -edge-cut C of G and suppose its capacity is greater than the total demand (else there is no solution). Any solution to the unsplittable flow problem solves a bin-packing problem where the demands are packed on the edges of C . If such a packing exists, it can be found by enumeration in time $O(\binom{k}{|C|}) = O(e^k)$. By a well-known Menger's theorem there exist $|C|$ edge-disjoint st -paths. We can route the unsplittable flow on these paths using the packing above. $\square$

4 First-Edge-Disjoint Paths

An instance of the first-edge-disjoint paths problem (FEDP) is a directed graph G and k pairs of terminals $s_1 t_1 \dots s_k t_k$. A *path allocation* is a k -tuple of paths from each source s_i to its corresponding sink t_i . A path allocation is *first-edge-disjoint* if in each path no first edge is shared with any other path in the path allocation. FEDP is to determine whether such a path allocation exists. It is easy to see that FEDP is NP-hard even if the underlying graph is acyclic, see the full version of this paper for a proof.³ In this section we will show that FEDP is fixed-parameter tractable.

² Recall that without the restriction on capacities the problem is $W[1]$ -hard on acyclic digraphs.

³ Similar but more complicated constructions show that on undirected and bi-directed graphs FEDP is NP-complete, too.

Call an edge e *blocked* in a path allocation ρ if it is the first edge of some $s_i t_i$ path in ρ . Given a set E_b of edges, we can decide in polynomial time if there is a first-edge-disjoint path allocation whose set of blocked edges is E_b . For each edge $s_i v \in E_b$ we can identify the set of sinks reachable from v in $E - E_b$. The rest is a bipartite matching problem: for each source node u , we want to match each source s_i located at u with some edge e from E_b that leaves u , such that t_i is reachable from the tail of e . Thus instead of looking for a complete path allocation, it suffices to identify a suitable set E_b of blocked edges. Note that checking all possible sets of blocked edges is not efficient since source nodes can have large out-degrees. We will show how to prune the search tree.

The first step of our algorithm is to convert the input to a standard form that better captures the structure relevant to the problem.

Definition 1. *An instance of FEDP is normal if the vertex set can be partitioned into three disjoint sets: $l \leq k$ source nodes $u_1 \dots u_l$, $l' \leq k$ sink nodes $v_1 \dots v_{l'}$ and a number of nonterminal nodes w_{ij} for each u_i . For each w_{ij} , there is an edge from u_i to w_{ij} . All other edges in the graph lead from nonterminals to terminals.*

In the full version of this paper we show that an FEDP instance can be converted, in time $O(mk)$, to an equivalent normal instance of size $O(mk)$. Henceforth we will assume that the FEDP instance is normal.

Definition 2. *Let k_i be the number of sources located at the node u_i . A terminal node r is i -accessible if there are at least $k_i + 1$ nonterminals w_{ij} such that there is an edge $w_{ij}r$. A terminal node r is i -blockable if there is an edge $w_{ij}r$ for some j but r is not i -accessible. A non-terminal node w_{ij} is i -easy if all nodes reachable from w_{ij} via a single edge are i -accessible. Otherwise, w_{ij} is i -hard.*

Call a path *blocked* if one of its edges is blocked. Note that if a terminal node v is i -accessible, then for any path allocation there is a non-blocked path from u_i to v via some nonterminal w_{ij} .

4.1 Reducing the Graph

Given an instance G of FEDP in the normal form, we construct an equivalent smaller instance G_R whose size depends only on k , such that G_R is a yes instance if and only if G is.

Consider first the special case when no two sources are located at the same node. Consider a source s_i located at the node u_i . Let T_i be the set of terminal nodes reachable in one step from some i -easy nonterminal. Let G' be an instance obtained from G by deleting all i -easy nonterminals and adding two new nonterminals w'_{i1} and w'_{i2} with edges from s_i to both w'_{i1} and w'_{i2} and edges from each of w'_{i1}, w'_{i2} to every node in T_i (note that the new nonterminals are i -easy). For any first-edge-disjoint path allocation ρ' in G' there is a first-edge-disjoint path allocation ρ in G . If in the path allocation ρ' the first edge of the $s_i t_i$ path goes to an i -hard node, the $s_i t_i$ path in ρ may use the same edge. If the $s_i t_i$ path in ρ' goes through one of the new nonterminals (let the next node on the path be some node r), the $s_i t_i$ path in ρ may use any i -easy nonterminal with an

edge to r . It is easy to check that this choice preserves the reachability relation between any pair of terminal nodes in G and G' .

Thus we may assume that there are at most two i -easy nodes for each s_i . Notice that for each s_i , there are at most $2k - 1$ i -hard nodes. Hence, for each source we have at most $2k$ choices for the first edge. The reduced graph has $O(k^3)$ edges, thus for each set of choices we can determine if it can be extended to a full path allocation in $O(k^4)$ time by depth first search. This reduction can be implemented in time $O(mk)$. Solving the reduced instance by enumeration gives us the following theorem.

Theorem 8. *If no two sources are located at the same node, the first-edge-disjoint paths problem can be solved in time $O(mk + k^4 (2k)^k)$.*

The reduction for the general case is similar. For each i , let T_i be the set of nodes reachable in one step from at least $k_i + 1$ i -easy nonterminals. For each i , we create $k_i + 1$ new nonterminals $w'_{i1}, \dots, w'_{i(k_i+1)}$ and add edges from u_i to each w'_{ij} and from every w'_{ij} to every node in T_i . Then we delete all edges from old w_{ij} nonterminals to vertices in T_i . Finally, we can delete all nonterminals without outgoing edges. As in the previous case, one can argue that the resulting graph G' is equivalent to the original. Consider source node u_i . There can be at most k_i edges entering each i -blockable terminal node r , there are $l + l' - 1$ terminals distinct from u_i , and hence there are at most $(k + l)k_i$ i -hard nonterminals. For each i -accessible terminal, there can be at most $k_i + 1$ edges entering it from i -easy nonterminals. Hence, there are no more than $2k(k_i + 1)$ i -easy nonterminals. Thus the reduced graph has $O(k^3)$ nodes. This reduction can be implemented in time $O(mk)$. Therefore FEDP is fixed-parameter tractable. A simple way to solve the reduced instance is to try all possible sets of blocked edges. In the rest of this subsection we give a more efficient search algorithm.

For a path allocation ρ let C_ρ be the set of i -hard nonterminals w_{ij} such that the edge $u_i w_{ij}$ is blocked in ρ , for all i . Suppose we are given C_ρ but not ρ itself. Then we can efficiently determine whether C_ρ was derived from a first-edge-disjoint path allocation.

Let E_ρ be the set of edges entering the nonterminals in C_ρ . Then, for each nonterminal w_{ij} , we can compute the set W_{ij} of terminal nodes r such that there is a path from w_{ij} to r in the graph $G - E_\rho$. Now we can formulate the following matching problem: for each source s_a , s_a located at node u_i , assign it an edge $u_i w_{ij}$ so that (a) each edge is assigned to at most one source, (b) $w_{ij} \in C_\rho$ or w_{ij} is i -easy, and (c) the sink t_a lies in W_{ij} .

Each first-edge-disjoint path allocation naturally defines a valid matching. It is easy to see that given a valid matching we can construct a first-edge-disjoint path allocation. Given a set C_ρ , the matching can be computed in $O(k^5)$ time using a standard Ford-Fulkerson max-flow algorithm.

We enumerate all possible sets C_ρ , and for each set check if it can be extended to a first-edge-disjoint path allocation. Recall that for each i , there are at most $x_i = (k + l - 1)k_i$ i -hard nonterminals. Hence, there are at most $\prod_{i=1}^l \sum_{j=1}^{k_i} \binom{x_i}{j}$ ways to choose the set C_ρ , which is $O((ek)^k)$ (see the full version of this paper). Therefore:

Theorem 9. *The first-edge-disjoint paths problem on directed graphs can be solved in time $O(mk + k^5 (ek)^k)$.*

In the full version of this paper we improve this running time for the case when the input graph is acyclic, and give a simple polynomial-time algorithm for the case when all sinks are located at the same node.

Acknowledgments. We thank Jon Kleinberg, Martin Pál and Mark Sandler for valuable discussions and help with the write-up.

References

1. G. Baier, E. Köhler and M. Skutella, "On the k -Splittable Flow Problem," *Proc. 10th Annual European Symposium on Algorithms*, 2002.
2. Y. Dinitz, N. Garg and M. Goemans "On the Single-Source Unsplittable Flow Problem," *Proc. 39th Annual Symposium on Foundations of Computer Science*, 1998.
3. R. Downey, V. Estivill-Castro, M. Fellows, E. Prieto and F. Rosamund, "Cutting Up is Hard to Do: the Parameterized Complexity of k -Cut and Related Problems," *Computing: The Australasian Theory Symposium*, 2003.
4. R.G. Downey and M.R. Fellows, *Parameterized Complexity*, Springer-Verlag (1999).
5. S. Even, A. Itai and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *SIAM J. Computing*, **5** (1976) 691-703.
6. S. Fortune, J. Hopcroft and J. Wyllie, "The directed subgraph homeomorphism problem," *Theoretical Computer Science*, **10** (1980) 111-121.
7. B. Korte, L. Lovász, H-J. Prömel, A. Schrijver, eds., *Paths, Flows and VLSI-Layouts*, Springer-Verlag (1990).
8. R.M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, R.E. Miller, J.W. Thatcher, Eds., Plenum Press, New York (1972) 85-103.
9. J. Kleinberg, "Single-source unsplittable flow," *Proc. 37th Annual Symposium on Foundations of Computer Science*, 1996.
10. —, "Decision algorithms for unsplittable flow and the half-disjoint paths problem," *Proc. 30th Annual ACM Symposium on the Theory of Computing*, 1998.
11. —, "Approximation Algorithms for Disjoint Paths Problems," *Ph.D. Thesis*, M.I.T, 1996.
12. S.G. Kolliopoulos and C. Stein, "Improved approximation algorithms for unsplittable flow problems," *Proc. 38th Annual Symposium on Foundations of Computer Science*, 1997.
13. C.L. Lucchesi and D.H. Younger, "A minimax relation for directed graphs," *J. London Mathematical Society* **17** (1978) 369-374.
14. N. Robertson and P.D. Seymour, "Graph minors XIII. The disjoint paths problem," *J. Combinatorial Theory Ser. B* **63** (1995) 65-110.
15. A. Schrijver, "Finding k disjoint paths in a directed planar graph," *SIAM J. Computing* **23** (1994) 780-788.
16. Y. Shiloach, "A polynomial solution to the undirected two paths problem," *J. of the ACM* **27** (1980) 445-456.
17. M. Skutella, "Approximating the single source unsplittable min-cost flow problem," *Mathematical Programming Ser. B* **91**(3) (2002) 493-514.
18. J. Vygen, "NP-completeness of some edge-disjoint paths problems," *Discrete Appl. Math.* **61** (1995) 83-90.
19. —, "Disjoint paths," *Rep. #94846*, Research Inst. for Discrete Math., U. of Bonn (1998).

Binary Space Partition for Orthogonal Fat Rectangles

Csaba D. Tóth

Department of Computer Science
University of California at Santa Barbara, CA 93106, USA,
`ttoth@cs.ucsb.edu`

Abstract. We generate a *binary space partition* (BSP) of size $O(n \log^8 n)$ and depth $O(\log^4 n)$ for n orthogonal fat rectangles in three-space, improving earlier bounds of Agarwal et al. We also give a lower bound construction showing that the size of an orthogonal BSP for these objects is $\Omega(n \log n)$ in the worst case.

1 Introduction

The *binary space partition* (BSP) is a data structure invented by the computer graphics community [9,6]. It was used for fast rendering polygonal scenes and for shadow generation. Ever since it found many application in computer graphics, robotics, and computational and combinatorial geometry.

A BSP is a recursive cutting scheme for a set of disjoint (or non-overlapping) polygonal scenes in the Euclidean space ($\mathbb{R}^3$). We split the bounding box of the polygons along a plane into two parts and then we partition recursively the two subproblems corresponding to the two subcells as long as the interior of a subcell intersects an input polygon. This partitioning procedure can be represented by a binary tree (called *BSP tree*) where every intermediate node stores a splitting plane and every leaf stores a convex cell. Similarly, the BSP can be defined for any set of $(d - 1)$ -dimensional objects in $\mathbb{R}^d$, $d \in \mathbb{N}$.

Two important parameters are associated to a BSP tree: The *size* $|P|$ of a BSP P is the set of nodes in the binary tree (note that $(|P| - 1)/2$ is the number of leaves of P , and the collection of leaves corresponds to a convex subdivision of the space). The *depth* of P is the length of the longest path starting from the root of the tree.

Combinatorial bounds on the size and the depth of the BSP in $\mathbb{R}^3$ were first obtained by Paterson and Yao [7] who showed that for n quadrilaterals in 3-space, there is always a BSP of size $O(n^2)$ and depth $O(\log n)$. This upper bound is asymptotically optimal in the worst case by a construction of Eppstein [4]. Research concentrated on finding polygonal scenes where a smaller BSP is possible: Paterson and Yao [8] proved that there exists a BSP of size $O(n^{3/2})$ for n disjoint orthogonal rectangles. They also provided a construction for a matching lower bound $\Omega(n^{3/2})$.

Agarwal et al. [1] generated a BSP of size $n2^{O(\sqrt{\log n})}$ and depth $O(\log n)$ for n disjoint orthogonal fat rectangles in three-space in $n2^{O(\sqrt{\log n})}$ time. A rectangle is *fat* (or α -*fat*) if its aspect ratio (the ratio of its longer and shorter edges) is bounded by a constant ($\alpha \in \mathbb{R}^+$). The main result of this paper improves upon the bound of Agarwal et al.:

Theorem 1. *For any set of n disjoint orthogonal fat rectangles in $\mathbb{R}^3$, there is a binary space partition of size $O(n \log^8 n)$ and depth $O(\log^4 n)$.*

Our proof is constructive and gives an explicit partitioning algorithm, where every splitting plane is axis-parallel. The main difference compared to the approach of Agarwal et al. [1] is that we exploit more geometric properties of fat rectangles, and this also makes the analysis of our algorithm considerably simpler. Our result implies that the function $n2^{O(\sqrt{\log n})}$ is not intrinsic to this problem (although it shows up miraculously in other geometric problems [11]). We also give a lower bound for *orthogonal BSPs* where all splitting planes are axis-parallel.

Theorem 2. *For any $n \in \mathbb{N}$, there are n disjoint orthogonal fat rectangles in $\mathbb{R}^3$ such that the size of any orthogonal BSP for it has size $\Omega(n \log n)$.*

Related results. De Berg [2] generated an $O(n)$ size BSP for n *full-dimensional* fat objects in $\mathbb{R}^d$ for any $d \in \mathbb{N}$ (in that case, the BSP is defined to partition space until every region intersects at most one object). His result does not apply to $(d-1)$ -dimensional fat polygonal objects in $\mathbb{R}^d$. Dumitrescu et al. [5] gave a tight bound, $\Theta(n^{5/3})$, on the worst case complexity of (not necessarily fat) orthogonal rectangles in $\mathbb{R}^4$.

In the plane, orthogonality or “fatness” can alone assure better upper bounds on the size of a BSP for n disjoint segments than the general $O(n \log n)$ bound of Paterson and Yao [7]. There is an $O(n)$ size BSP if the ratio of the longest and the shortest segment is bounded by a constant [3], or if the segments have a constant number of distinct orientations [10].

2 Preliminaries

Notation. We call an axis parallel open box in $\mathbb{R}^3$ a *cell*. The bounding box of the fat rectangles is the initial cell for our partitioning algorithm, and every region corresponding to a node of an orthogonal BSP is also a cell.

Definition 1. *Given a cell C and an orthogonal rectangle r intersecting the interior of C , we say that*

- r is long with respect to C , if no vertex of r lies in the interior of C ;
- r is a free-cut for C if none of the edges of r intersects the interior of C .
- r is a shelf for C if exactly one edge of r intersects the interior of C .
- r is a bridge for C if exactly two parallel edges of r intersect $\text{int}(C)$.

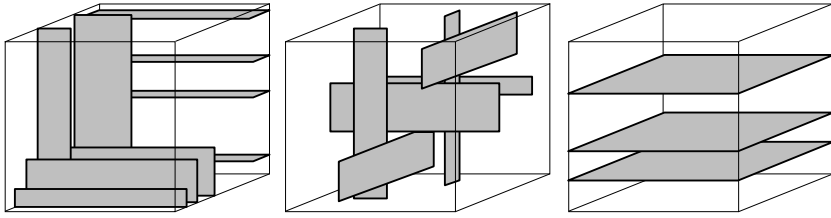


Fig. 1. Shelves, bridges, and free-cuts for a cell C .

Free-cuts, shelves, and bridges for a cell C are also long w.r.t. C . Our partitioning algorithm will make use of every free-cut, even if it is not always stated explicitly: Whenever a fragment $r \cap C$ is a free-cut for an intermediate cell C of the partitioning, we partition C along $r \cap C$. Therefore we may assume that every long rectangle w.r.t. a cell C is either a bridge or a shelf.

The distinction between shelves and bridges allows us to exploit the geometric information they carry. The aspect ratio of the clipped bridge $r \cap C$ is constraint if r is an α -fat rectangle: The length of the edge of $r \cap C$ in the interior of C is at most α times bigger than its edge along the boundary of C (we refer to this property as “*semi-fatness*”). The intersection $r \cap C$ of a shelf r and the cell C can have arbitrary aspect ratio, but it has only one edge in the interior of C (in particular, our Lemma 2 shows that there is an $O(n \log n)$ size BSP for n shelves in a cell).

A line segment (resp., edge of a rectangle) is called x -, y -, or z -segment (resp., x -, y -, or z -edge) if it is parallel to the corresponding coordinate axis. The *orientation* of an orthogonal rectangle is the pair of orientations of its edges, thus we talk about xy -, yz -, and xz -rectangles. The *base* of a shelf $r \cap C$ in C is the side of C containing the edge of $r \cap C$ opposite to the edge in the interior of C . All shelves for a side s of C must have the same orientation, because the orientation of a shelf is different from that of s and two shelves for s with the remaining two distinct orientations cannot be disjoint.

Agarwal et al. [1] distinguish three *classes* of long (but not free-cut) rectangles w.r.t. C : A clipped long rectangle $r \cap C$ belongs to the x -class (y -class, z -class) if the edge of r in $\text{int}(C)$ is a x -edge (y -edge, z -edge). They have shown that there is a BSP of $O(n)$ size and $O(n)$ depth (note that there is also a BSP of $O(n \log n)$ size and $O(\log n)$ depth) for n rectangles which are all long w.r.t. C . This BSP, however, does not provide a good partitioning for all the rectangles, because it possibly partitions a rectangle in the interior of C into $O(n)$ pieces.

Overlay of BSPs. The powerful method to construct BSPs is a combination of several BSPs, that we call an *overlay BSP*. It allows us to partition the same region several times such that the size of the overlay BSP is no more than the sum of the BSPs used. Consider the following setting: We are given a set F of objects in a cell C , a BSP P for F , and a subdivision $S(C)$ of C (which may be a result of a BSP for some other set of objects F' in C). We refine recursively

the subdivision $S(C)$ according to the cuts made by the BSP P for F : Whenever P splits a subcell $C' \subset C$ into two parts, we split every region R of the current subdivision which lies along the splitting plane if the interior of R intersects an object of F and the splitting plane actually divides $F \cap R$. If F contains no objects in R or all objects of $F \cap R$ lie on one side of the splitting plane, then we keep R intact.

Fig. 2 illustrates the overlay on a planar example. F is a set of disks in a square C , $S(C)$ is an orthogonal subdivision of C (Fig. 2, left). A BSP P for F is depicted in the middle. On the right of Fig. 2, bold segments indicate the portions of splitting lines of P which are used in the overlay to refine the subdivision $S(C)$.

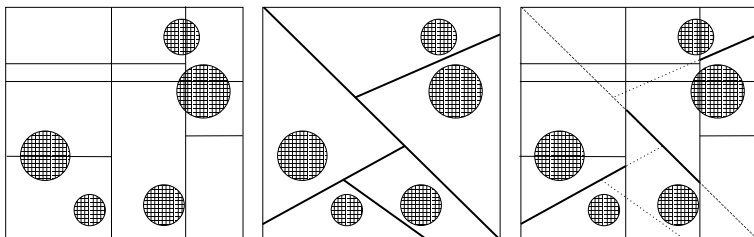


Fig. 2. A subdivision, a BSP for the disks, and their overlay.

If we have k independent BSPs $P_1, P_2, \dots, P_k$ for sets $F_1, F_2, \dots, F_k$ of disjoint objects in the same region C , then we can obtain their overlay by starting with a trivial subdivision $S_0 = \{C\}$ and recursively forming the overlay of S_{i-1} and P_i ($i = 1, 2, \dots, k$) to obtain a refined subdivision S_i . The resulting overlay partitioning is a BSP (since a splitting line of P_i splits –simultaneously– regions of S_i into two parts), and it is a BSP for each of $F_1, F_2, \dots, F_k$. The depth of the overlay BSP is no more than the sum of the depth of $P_1, P_2, \dots, P_k$. If each of $P_1, P_2, \dots, P_k$ cuts every line segment at most ℓ times, then the overlay BSP partitions every line segment at most $k\ell$ times.

Computation of BSP size. To compute the size of a BSP P , we will count the total number $c(P)$ of fragments of the objects obtained by the partitioning. $2 \cdot c(P)$ gives, in turn, an upper bound on the size of P if we suppose that P makes *useful cuts* only. A useful cut means that whenever P splits a subcell D into D_1 and D_2 then either the splitting plane lies along a $((d-1)$ -dimensional) object or both $\text{int}(D_1)$ and $\text{int}(D_2)$ intersect some objects. Thus every useful cut partitions $F \cap \text{int}(D)$. Since eventually every fragment lies on one of the splitting planes, $2 \cdot c(P)$ is an upper bound on the number of nodes in the binary tree. So we obtain a bound on the size of a BSP P , if we know how many fragments each of the fat rectangles are partitioned into. In our analysis, we will concentrate on how a 1-dimensional object is fragmented:

Definition 2. Given a set F of pairwise disjoint fat orthogonal rectangles, a *mast* is an axis-parallel line segment within a rectangle of F .

Remark 1. Suppose that an orthogonal BSP P dissects any *mast* into at most k pieces. This implies that P partitions every orthogonal fat rectangle into at most k^2 fragments. Since we exploit every free-cut right after they are created, the number of fragments of the rectangle in disjoint subcells is at most $4k - 4$.

3 Main Loop of Partition Algorithm

Lemma 1. Given an axis-parallel cell C , a set F of n fat orthogonal rectangles, and a subset L , $L \subseteq F$, of rectangles long w.r.t. C , there is an orthogonal BSP of depth $O(\log^3 n)$ for L such that it partitions every *mast* into $O(\log^3 n)$ pieces.

Note that rectangles of F can possibly be long w.r.t. a subcell C' formed by the BSP. However no fragment of a rectangle which is long w.r.t. C intersects a subcell after this BSP. The proof of this lemma is postponed to the next section. Here we present an algorithm (using the BSP claimed by Lemma 1 as a subroutine) that establishes our main theorem.

Algorithm 1 *Input: set F of orthogonal fat rectangles and the bounding box C .*

1. Initialize $i = 0$, $\mathcal{C}_0 = \{C\}$, $S_0 = \{C\}$, and let V denote the vertex set of all rectangles in F .
2. While there is a cell $C \in \mathcal{C}_i$ which contains a vertex of V in its interior, do
 - a) For every cell $C_i \in \mathcal{C}_i$ where $V \cap \text{int}(C) \neq \emptyset$, split C_i into eight pieces by the three axis-parallel medians of the point set $V \cap \text{int}(C_i)$.
 - b) For all $C_i \in \mathcal{C}_i$, let $\mathcal{C}_{i+1}$ denote the collection of these eight pieces.
 - c) In every subcell $C' \in \mathcal{C}_{i+1}$, compute a BSP $P_L(C')$ for the set of rectangles which are long w.r.t. C' according to Lemma 1; and form the overlay BSP of $S_i \cap C'$ and $P_L(C')$. Let $S'_i(C')$ denote the refined subdivision.
 - d) Let S_{i+1} be the union of the subdivisions $S'_i(C')$, $C' \in \mathcal{C}_{i+1}$. Set $i := i + 1$.

Proof (of Theorem 1). We call a *round* the work done by the algorithm between increments of i . Notice that the algorithm is completed in $\log(4n) = O(\log n)$ rounds because step 2a decreases the number of rectangle vertices in a cell (originally $4n$) by a factor of two.

Consider a *mast* e (c.f. Definition 2). We argue that e is partitioned into $O(\log^4 n)$ fragments throughout the algorithm. Step 2a and 2c can only dissect a fragment $e \cap C_i$ of the *mast* if the rectangle fragment $r \cap C_i$ is incident to one of the four vertices of r ; otherwise $r \cap C_i$ is long w.r.t. C_i and it was eliminated in a step 2c of an earlier round.

Let $e = uv$ and suppose that up to round i steps 2a dissected e at points $w_1, w_2, \dots, w_k$ (this does not include cuts made by step 2c). In round $i + 1$, only the fragments uw_1 and w_kv can be further partitioned. Therefore in one round, step 2a can cut e at most twice. Then step 2c can cut e at most $4O(\log^3 n) =$

$O(\log^3 n)$ times: Both halves of uw_1 and w_kv are partitioned into $O(\log^3 n)$ times by the overlay BSP. In the course of $O(\log n)$ rounds, e is dissected $O(\log^4 n)$ times. In sight of Remark 1 this means that every fat rectangle is cut into $O(\log^8 n)$ fragments, and the size of the resulting BSP is $O(n \log^8 n)$. $\square$

4 Building Blocks

In this section we prove Lemma 1. Our partitioning scheme is build of three levels of binary partitions. We discuss each level in a separate subsection below. First we show how to find a BSP for shelves while cutting masts at most $O(\log n)$ times in Subsection 4.1. We next aim at eliminating the bridges for a given cell C . In Subsection 4.2, we reduce the problem to long rectangles of one class only. Then in Subsection 4.3 we describe a BSP for long rectangles in one class. The arguments follow the lines of the main theorem.

4.1 BSP for Shelves

In this subsection we prove the following lemma which serves then as a basic building block for the other two levels of partitions.

Lemma 2. *Given an axis-parallel cell C , a set F of fat orthogonal rectangles, and a set L of ℓ shelves for C . There is a BSP of depth $O(\log(\ell + 1))$ for L such that it partitions every mast into $O(\log(\ell + 1))$ pieces.*

Note that the resulting subcells in this BSP can have shelves, but those rectangles were not shelves w.r.t. the initial cell C . The lemma states only that fragments of every shelf w.r.t. C do not intersect the interior of any resulting subcell.

Here, we state and prove a simplified version of Lemma 2, Lemma 3, which together with the concept of overlay will imply Lemma 2.

Lemma 3. *Given an axis-parallel cell C , a set F of fat orthogonal rectangles, and a set L of ℓ shelves of one sides of C . There is a BSP of depth $O(\log(\ell + 1))$ for L such that it cuts every mast into $O(\log(\ell + 1))$ pieces.*

Proof. We may assume without loss of generality that the lower xz -side of C is the base of all shelves in L and the orientation of every shelf in L is yz (see Fig. 3). Notice that the x -coordinate of every shelf is different.

Algorithm 2 *The input is a pair (C, L) where C is an axis-parallel cell and L is a set of yz -shelves for the lower xz -side of C . Let y_0 be the highest y -coordinate of all shelves, and let x_0 be the median x -coordinate of the shelves.*

1. Dissect C by the plane $y = y_0$.
2. Dissect the part below $y = y_0$ by the plane $x = x_0$.
3. Make a free-cut along the shelf with highest y -coordinate.
4. Call recursively this algorithm for $(C', L \cap C')$ in every subcell C' where $L \cap \text{int}(C')$ is non-empty.

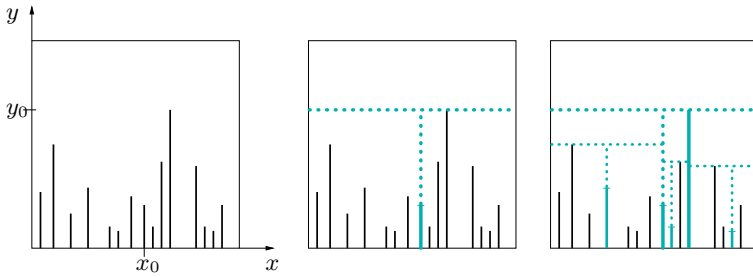


Fig. 3. Two consecutive rounds of Algorithm 2 on shelves.

Let us call the work done between recursive calls of Algorithm 2 a *round*. In one round, C is cut into four pieces. The portion of C above the plane $y = y_0$ is disjoint from L . Each of the three parts below $y = y_0$ contains less than half as many shelves of L as C . Therefore the algorithm terminates in $O(\log(\ell + 1))$ rounds.

The algorithm does not cut any z -segments. It dissects any y -segment into $O(\log(\ell + 1))$ pieces, because it can be cut at most once in every level of the recursion. Let e be a mast of direction x . Observe that a fragment $e \cap C'$ in a subcell C' can only be cut if C' contains an endpoint of e in its interior. Otherwise e lies above the highest shelf in C' , because e is clipped to a fat rectangle and therefore it is disjoint from the shelves. This implies that e is cut at most four times in each round, and so it is partitioned into $O(\log(\ell + 1))$ pieces during the algorithm. $\square$

Proof (of Lemma 2). Let $P_1, P_2, \dots, P_6$ be the six BSPs obtained by applying Lemma 3 independently for the shelves of of the six sides of C . We create the overlay P_σ of these six BSPs. Each of $P_1, P_2, \dots, P_6$ partitions every mast e of a rectangle r into $O(\log(\ell + 1))$ pieces. Therefore, P_σ also partitions e into $6 \cdot O(\log(\ell + 1)) = O(\log(\ell + 1))$ pieces. $\square$

4.2 Reduction to Shelves and One Class of Long Rectangles

Lemma 4. *Given an axis-parallel cell C , a set F of n fat orthogonal rectangles, and a subset L of rectangles long w.r.t. C . One can recursively dissect C by axis-parallel planes such that in each resulting subcell C' the rectangles which intersect the interior of C' and are long w.r.t. C belong to the one class; the depth of the partitioning is $O(\log^2 n)$ and every mast is cut into $O(\log^2 n)$ pieces.*

Assume we are given a cell C and a set L of long rectangles from two or three classes. We describe a partitioning algorithm and prove Lemma 4. As a first step, we reduce the problem to two classes of long rectangles (long w.r.t. C) intersecting each subcell. We may assume w.l.o.g. that the x -edge and the z -edge of C is the longest and the shortest edge of C respectively.

Dissect C into $(\lfloor \alpha \rfloor + 1)^2$ congruent cells by dividing C with $\lfloor \alpha \rfloor, \lfloor \alpha \rfloor$ equally spaced xy -planes and xz -planes. In the following proposition, we use the “semi-fatness” of the bridges.

Proposition 1. *In each of the $(\lfloor \alpha \rfloor + 1)^2$ subcells C' of C , a fragment $f \in L \cap C'$ is either*

- (i) *a shelf for C' ; or*
- (ii) *a bridge for C' in the z -class; or*
- (iii) *a bridge for C' in the y -class with orientation xy .*

Proof. Consider a subcell C' . Since the x -edge of C' is more than α times longer than its y - and z -edge, there are no bridge for C' in the x -class. Similarly, the y -edge of C' is more than α times longer than the z -edge of the initial cell C , therefore a bridge for C in the class y with orientation yz cannot be a bridge for C' . $\square$

Next we apply Lemma 2 in each of the $(\lfloor \alpha \rfloor + 1)^2$ subcells to eliminate the shelves for those subcells. The $(\lfloor \alpha \rfloor + 1)^2$ subdivisions together can cut every mast into $(\lfloor \alpha \rfloor + 1) \cdot O(\log n) = O(\log n)$ pieces. We obtain $(\lfloor \alpha \rfloor + 1)^2$ congruent subcells C' with a subdivision $S(C')$ where fragments of L are in case (ii) or (iii) of Proposition 1. The overlay of any further BSP with the subdivision $S(C')$ (or with its refinement) will not cut any shelf for C' .

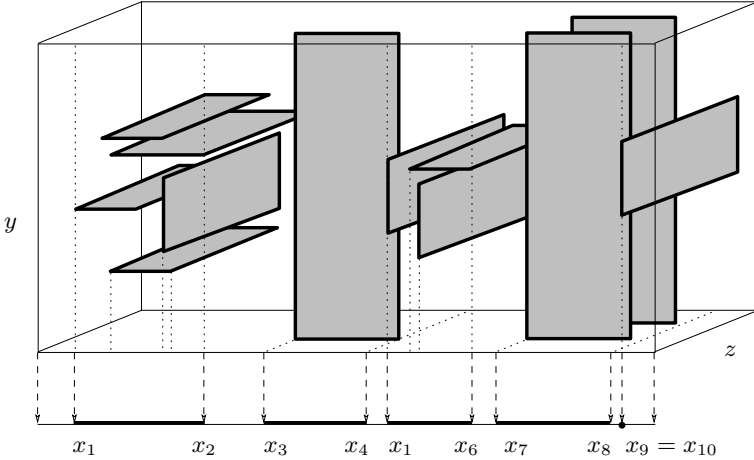


Fig. 4. Bridges from two classes.

Now consider a cell C where every long rectangle w.r.t. C belongs to the y -class with orientation xy or to the z -class (see Fig. 4). Project every long rectangle r w.r.t. C to the x axis. Since the projections of bridges from different classes do not overlap, we can cover the projections with disjoint intervals

$x_1x_2, x_3x_4, \dots, x_{k-1}x_k$ such that each interval covers (projections of) rectangles from the same class. Let $\bar{x}(L \cap C) = (x_1, x_2, \dots, x_k)$ be the sequence of interval endpoints.

If we cut C by yz -planes through $x_1, x_2, \dots, x_k$, then each piece would satisfy Lemma 4, but some other rectangles could be cut k times. Fortunately this may only happen to shelves for C , as we show in the following proposition.

Proposition 2. *A rectangle in the interior of C (i.e., which is disjoint from the boundary of C) intersects at most $4\alpha + 2$ consecutive planes from the set of yz -planes $\{x = x_1, x = x_2, \dots, x = x_k\}$.*

Proof. We have assumed that the y -edge of C' is at least as long as its z -edge. We also know that every bridge in the y -class has orientation xy . Therefore the interval corresponding to bridges of the y -class is at least $1/\alpha$ times as long as the y - and the z -edge of C' . This implies that an α -fat rectangle f which lies completely in the interior of C' intersects at most α sections of y -class bridges. Since every second interval corresponds to y -class bridges, f intersects at most $2\alpha + 1$ intervals. $\square$

Let $\bar{\bar{x}}(L \cap C) = (x_{6\lceil\alpha\rceil}, x_{12\lceil\alpha\rceil}, \dots, x_{\lfloor k/6 \rfloor \cdot 6\lceil\alpha\rceil})$ be the subsequence of $\bar{x}$ containing every $6\lceil\alpha\rceil$ -th element ($\bar{\bar{x}}$ is empty if $|\bar{x}| < 6\lceil\alpha\rceil$), and let $\hat{x}(L \cap C)$ denote the median of $\bar{\bar{x}}$. We will cut along the planes $x = x_0, x_0 \in \bar{\bar{x}}$, in a binary order, and add clean-up steps in between to save shelves.

Algorithm 3 *Input: $(C, L, S(C))$ where C is a cell, L is a set of long rectangles w.r.t. C which are either y -class xy -oriented or in the z -class, and $S(C)$ is an orthogonal subdivision of C .*

1. If $\bar{\bar{x}}(L \cap C)$ is empty then cut C by the planes $x = x_i, x_i \in \bar{x}(L \cap C)$ and exit.
2. Otherwise, do:
 - a) Dissect C by $x = \hat{x}(L \cap C)$ into C_1 and C_2 .
 - b) For $i = 1, 2$, compute a BSP $P_S(C_i)$ for the set of shelves for C_i according to Lemma 2; and form the overlay BSP of $S(C) \cap C_i$ and $P_S(C_i)$. Let $S(C_i)$ denote the refined subdivision.
 - c) Call recursively this algorithm for $(C_1, L \cap C_1, S(C_1))$ and for $(C_2, L \cap C_2, S(C_2))$.

Proof (of Lemma 4). In every round (i.e., recursive call) of Algorithm 3, the number of elements in the sequence $\bar{x}(L \cap C)$ is halved, so the algorithm terminates in $O(\log n)$ rounds.

Consider a mast e parallel to the x -axis. As long as $\bar{\bar{x}}$ is non-empty, a plane $x = \hat{x}$ can partition a fragment $e \cap C'$ in step 2a if C' contains one of the endpoints of e . If a fragment $e \cap C'$ is long w.r.t. the cell C' then it is a shelf for C' by Proposition 2 and it has been eliminated in a step 2b.

As long as $|\bar{x}(L \cap C)| \geq 6$, step 2a cuts e at most twice in every round, and step 2b cuts it $4O(\log n) = O(\log n)$ times. Once the cardinality of $\bar{x}(L \cap C)$ drops below 6, the two fragments of e incident to the endpoints of e can be further cut

five more times (a total of ten more cuts) by planes $x = x_i$. Therefore in course of $O(\log n)$ rounds the mast e is cut into $O(\log^2 n)$ pieces.

Now consider a mast e parallel to the y - or z -axis. e is not cut by any plane $x = x_0$, $x_0 \in \bar{x}(L \cap C)$. It lies in the overlay of $O(\log n)$ shelf-partitions, and therefore it is cut into $O(\log^2 n)$ fragments. $\square$

4.3 BSP for One Class of Long Rectangles

Lemma 5. *Given an axis-parallel cell C , a set F of n fat orthogonal rectangles, and a subset L of long rectangles w.r.t. C in the x -class. There is a BSP of depth $O(\log^4 n)$ for L such that it partitions every edge into $O(\log^4 n)$ pieces.*

Algorithm 4 *Input: $(C, L, V, S(C))$ where C is a cell, L is a set of x -class long rectangles w.r.t. C , V is subset of vertices of L , and $S(C)$ is an orthogonal subdivision of C .*

1. Split C into four pieces C_1, C_2, C_3, C_4 by the two medians of the point set $V \cap \text{int}(C_i)$ of orientation xy and xz .
2. In C_i , $i = 1, 2, 3, 4$, compute a BSP $P_O(C_i)$ which partitions the long rectangles w.r.t. C_i such that every subcell contains one class of long rectangles w.r.t. C_i according to Lemma 4; and form the overlay BSP of $S(C) \cap C_i$ and $P_O(C_i)$. Let $S(C_i)$ denote the refined subdivision.
3. Call recursively this algorithm for every $(C_i, L \cap C_i, V \cap C_i, S(C_i))$ where $L \cap C_i$ is non-empty.

Proof (of Lemma 5). We call Algorithm 4 with input C , L , and letting V be the set of all vertices of all rectangles in L and $S(C) := \{C\}$. First note that the algorithm terminates in $\log n$ rounds (recursive calls), because in every round step 1 halves the the number of vertices of V lying in the interior of a cell C .

Consider a mast e parallel to the y - or z -axis. A fragment $e \cap C'$ can be cut if C' contains an endpoint of e in its interior. Otherwise e is clipped to a rectangle r where $r \cap C'$ is long w.r.t. C' in the y -class or in the z -class and therefore $r \cap C'$ was separated from elements of L in step 2 of an earlier round. That means that $e \cap C'$ is not partitioned any further by overlays of a BSP for L . Therefore in one round, step 1 can cut e twice and step 2 can cut it $4 \cdot O(\log^2 n) = O(\log^2 n)$ times. During $O(\log n)$ rounds of the algorithm, e is dissected into $O(\log^3 n)$ pieces.

Finally, a z -mast is never cut by step 1. It lies in the overlay of $O(\log n)$ BSPs obtained by Lemma 4, and so it is dissected into $O(\log^3 n)$ pieces. $\square$

Proof (of Lemma 1). First we subdivide C such that every subcell contains at most one class of long rectangles from L (Lemma 4). This subdivision already eliminates all the shelves of L (by repeated use of Lemma 2). Then we eliminate all the bridges from L by Lemma 5. The complexity of this BSP is asymptotically the same as that of Algorithm 4. $\square$

5 Lower Bound

We describe two families of k squares in $\mathbb{R}^3$ (see Fig. 5 for an illustration). A square is given by six coordinates: three-three coordinates of two opposite vertices:

$$G(k) = \{g_i = [(i - k, 0, i), (i, k, i)] : i = 1, 2, \dots, k\},$$

$$H(k) = \left\{ h_i = \left[\left(i + \frac{1}{2}, k - \frac{i}{2}, i + \frac{1}{2} \right), \left(i - k + \frac{1}{2}, k - \frac{i}{2}, i - k + \frac{1}{2} \right) \right] : \right. \\ \left. i = 1, 2, \dots, k \right\}.$$

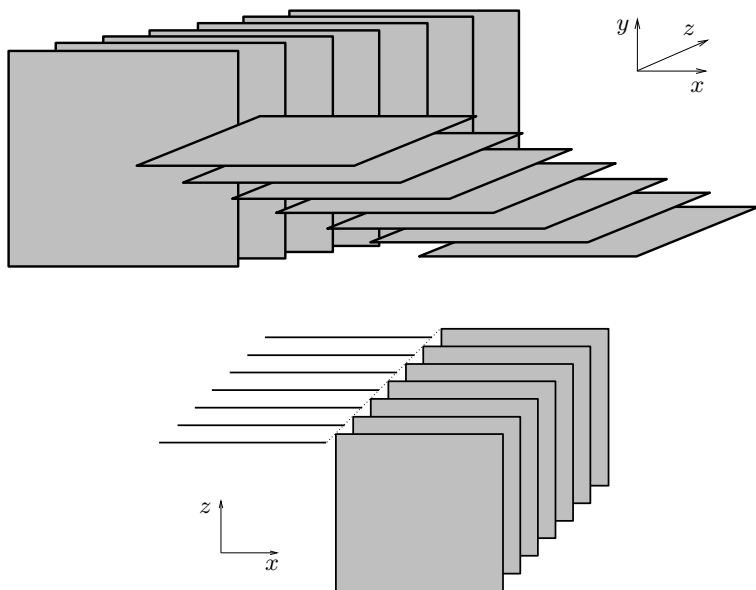


Fig. 5. $F(7, 0) = G(7) \cup H(7)$ in perspective view (left) and in top view (right).

The construction $F(k, \ell)$ is the union of $G(k)$, $H(k)$, and ℓ horizontal rectangles under h_k . We show that any orthogonal BSP for $F(k, \ell)$ has size $\Omega(k \log k)$. This implies that any orthogonal BSP for $F(k, 0)$ with $2k$ fat orthogonal rectangles has size $\Omega(k \log k)$. It is sufficient to write up a recursion formula considering the cases that $F(k, \ell)$ is partitioned by a plane of three different orientation.

An xy -plane $z = j$, $j \in \{1, 2, \dots, k\}$, through g_j cuts $j + \ell$ horizontal rectangles and dissects $F(k, \ell)$ into an $F(j - 1, \ell)$ and an $F(k - j, \ell + j)$.

A yz -plane $x = j$, $j \in \{1, 2, \dots, k\}$, cuts j rectangles from $H(k)$ and $k - j$ rectangles from $G(k)$. It dissects $F(k, \ell)$ into an $F(j, 0)$ and an $F(k - j, \ell + j)$.

An xz -plane $xy = j - 1/2$, $j \in \{1, 2, \dots, k\}$, through h_j cuts all k rectangles from $G(k)$; and it dissects $F(k, \ell)$ into an $F(j - 1, 0)$ and an $F(k - j, \ell)$.

Denoting by $f(k, \ell)$ the minimum number of cuts made by a BSP for $F(k, \ell)$, we have $f(1, \ell) = 0$ and the recursion formula

$$f(k, \ell) \geq \min \{ \min_j f(j-1, \ell) + f(k-j, \ell+j) + j + \ell, \quad (1)$$

$$\min_j f(j, 0) + f(k-j, \ell+j) + k, \quad (2)$$

$$\min_j f(j-1, 0) + f(k-j, \ell) + k \}. \quad (3)$$

$$f(1, \ell) = 0. \quad (4)$$

6 Conclusion

For every set of n disjoint fat orthogonal rectangles in three-space, there is an orthogonal BSP of size $O(n \log^8 n)$ and depth $O(\log^4 n)$. This improves an earlier bound of $n2^{O(\sqrt{\log n})}$ of Agarwal et al. We have seen that an $O(n \text{ polylog } n)$ bound is best possible for orthogonal BSPs.

The true complexity of a BSP for orthogonal squares remains unknown. Future work can focus on proving a super-linear lower bound on the size of a (generic) BSP for such objects. It is possible that the steps of our partitioning algorithm can be organized in a more intriguing fashion so that they yield a better upper bound (i.e., a smaller exponent on the logarithmic factor).

References

1. AGARWAL, P. K., GROVE, E. F., MURALI, T. M., AND VITTER, J. S.: Binary space partitions for fat rectangles. *SIAM J. Comput.* **29** (2000), 1422–1448.
2. DE BERG, M.: Linear size binary space partitions for uncluttered scenes. *Algorithmica* **28** (3) (2000), 353–366.
3. DE BERG, M., DE GROOT, M., AND OVERMARS, M.: New results on binary space partitions in the plane. *Comput. Geom. Theory Appl.* **8** (1997), 317–333.
4. DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O.: *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.
5. DUMITRESCU, A., MITCHELL, J. S. B., AND SHARIR, M.: Binary space partitions for axis-parallel segments, rectangles, and hyperrectangles. In *Proc. 17th ACM Symp. on Comput. Geom.* (Medford, MA, 2001), ACM Press, pp. 141–150.
6. FUCHS, H., KEDEM, Z. M., AND NAYLOR, B.: On visible surface generation by a priori tree structures. *Comput. Graph.* **14** (3) (1980), 124–133. Proc. SIGGRAPH.
7. PATERSON, M. S., AND YAO, F. F.: Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.* **5** (1990), 485–503.
8. PATERSON, M. S., AND YAO, F. F.: Optimal binary space partitions for orthogonal objects. *J. Algorithms* **13** (1992), 99–113.
9. SCHUMACKER, R. A., BRAND, R., GILLILAND, M., AND SHARP, W.: Study for applying computer-generated images to visual simulation. Tech. Rep. AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory, 1969.
10. TÓTH, Cs. D.: Binary space partition for line segments with a limited number of directions, *SIAM J. Comput.* **32** (2) (2003), 307–325.
11. TÓTH, G.: Point sets with many k -sets. *Discrete Comput. Geom.* **26** (2) (2001), 187–194.

Sequencing by Hybridization in Few Rounds

Dekel Tsur

Dept. of Computer Science, Tel Aviv University
dekelts@tau.ac.il

Abstract. Sequencing by Hybridization (SBH) is a method for reconstructing an unknown DNA string based on substring queries: Using hybridization experiments, one can determine for each string in a given set of strings, whether the string appears in the target string, and use this information to reconstruct the target string. We study the problem when the queries are performed in rounds, where the queries in each round depend on the answers to the queries in the previous rounds. We give an algorithm that can reconstruct almost all strings of length n using 2 rounds with $O(n \log_\alpha n / \log_\alpha \log_\alpha n)$ queries per round, and an algorithm that uses $\log_\alpha^* n - \Omega(1)$ rounds with $O(n)$ queries per round, where α is the size of the alphabet.

We also consider a variant of the problem in which for each substring query, the answer is whether the string appears once in the target, appears at least twice in the target, or does not appear in the target. For this problem, we give an algorithm that uses 3 rounds of $O(n)$ queries. In all our algorithms, the lengths of the query strings are $\Theta(\log_\alpha n)$. Our results improve the previous results of Margaritis and Skiena [17] and Frieze and Halldórsson [10].

1 Introduction

Sequencing by Hybridization (SBH) [4, 16] is a method for sequencing of long DNA molecules. In this method, the target string is hybridized to a chip containing known strings. For each string in the chip, if its reverse complement appears in the target, then the two strings will bind (or hybridize), and this hybridization can be detected. Thus, SBH can be modeled as the problem of finding an unknown target string using queries of the form “Is S a substring of the target string?” for some string S . Classical SBH consists of making queries for all the strings of length k for some fixed k , and then constructing the target string using the answers to the queries.

Unfortunately, string reconstruction is often not unique: Other strings can have the same spectrum as the target’s. Roughly, for an alphabet of size α , only strings of length about $\alpha^{\frac{1}{2}k}$ can be reconstructed reliably when using queries of length k [20, 8, 3, 22]. In other words, in order to reconstruct a string of length n , it is required to take $k \approx 2 \log_\alpha n$, and thus the number of queries is $\Theta(n^2)$. As this number is large even for short strings, SBH is not considered competitive in comparison with standard gel-based sequencing technologies.

Several methods for overcoming the limitations of SBH were proposed: alternative chip designs [20, 9, 21, 13, 14, 11, 15], using location information [1, 6, 12, 5, 7, 22], using a known homologous string [19, 18, 26], and using restriction enzymes [25, 23].

Margaritis and Skiena [17] suggested asking the queries in several rounds, where the queries in each round depend on the answers to the queries in the previous rounds. The goal is to design algorithms that use as few rounds as possible, and each round contains as few queries as possible. Margaritis and Skiena [17] gave several results, including an algorithm for reconstructing a random string of length n with high probability in $O(\log_\alpha n)$ rounds, where the number of queries in each round is $O(n)$. They also gave several worst-case bounds: For example, they showed that every string of length n can be reconstructed in $O(\log n)$ rounds using $n^2 / \log n$ queries in each round. Skiena and Sundaram [24] showed that every string can be reconstructed in $(\alpha - 1)n + O(\sqrt{n})$ rounds with one query per round. They also showed that at least $\frac{1}{4}(\alpha - 3)n$ queries are needed in the worst-case. Frieze et al. [9] showed that in order to reconstruct a random sequence with constant success probability, $\Omega(n)$ queries are needed.

Frieze and Halldórsson [10] studied a variant of the problem, in which for each substring query, the answer is whether the string appears once in the target, appears at least twice in the target, or does not appear in the target. We call this model the *ternary spectrum model*, while the former model will be called the *binary spectrum model*. For the ternary spectrum model, Frieze and Halldórsson gave an algorithm that uses 7 rounds with $O(n)$ queries in each round.

In this paper, we improve the results of Margaritis and Skiena, and of Frieze and Halldórsson. For the binary spectrum model, we give an algorithm that can reconstruct a random string with high probability in 2 rounds using $O(n \log_\alpha n / \log_\alpha \log_\alpha n)$ queries per round, and an algorithm that reconstruct a random string with high probability in $\log_\alpha^* n - c$ rounds using $O(n)$ queries per round, for every constant c (the constant hidden in the bound on the number of queries in each round depends on c). For the ternary spectrum model, we give an algorithm that can reconstruct a random string with high probability in 3 rounds using $O(n)$ queries per round. In addition to improving the number of rounds, our analysis of the latter algorithm is simpler than the analysis of Frieze and Halldórsson.

The rest of this paper is organized as follows: Section 2 contains basic definitions and top-level description of our algorithms. In Section 3 we give the algorithms for the binary spectrum model, and in Section 4 we give the algorithm for the ternary spectrum model.

2 Preliminaries

For clarity, we shall concentrate on the case of alphabet of size 4, which is the alphabet size of DNA strings (specifically, let $\Sigma = \{A, C, G, T\}$). However, our results hold for every finite alphabet.

For a string $A = a_1 \cdots a_n$, let A_l^i denote the l -substring $a_i a_{i+1} \cdots a_{i+l-1}$. The *binary k -spectrum* of a string A is a mapping $\text{SP}_2^{A,k}: \Sigma^k \rightarrow \{0, 1\}$ such that $\text{SP}_2^{A,k}(B) = 1$ if B is a substring of A , and $\text{SP}_2^{A,k}(B) = 0$ otherwise. The *ternary k -spectrum* of A is a mapping $\text{SP}_3^{A,k}: \Sigma^k \rightarrow \{0, 1, 2\}$, where $\text{SP}_3^{A,k}(B) = 0$ if B is not a substring of A , $\text{SP}_3^{A,k}(B) = 1$ if B appears in A exactly once, and $\text{SP}_3^{A,k}(B) = 2$ if B appears in A twice or more. We shall omit the subscript when referring to a spectrum of unspecified type, or when the type of the spectrum is clear from the context.

Let $\log_a^{(1)} n = \log_a n$ and $\log_a^{(i)} n = \log_a(\log_a^{(i-1)} n)$ for $i > 1$. Define $\log_a^* n$ to be the minimum integer i such that $\log_a^{(i)} n \leq 1$. When omitting the subscript, we shall assume base 4.

In the following, we say that an event happens with high probability (w.h.p.) if its probability is $1 - n^{-\Omega(1)}$.

Let $A = a_1 \cdots a_n$ denote the target string. All our algorithms have the same basic structure:

1. $k \leftarrow k_0$.
2. Let $Q = \{x_1 x_2 \cdots x_k : x \in \Sigma\}$. Ask the queries in Q and construct $\text{SP}^{A,k}$.
3. For $t = 1, \dots, T$ do:
 - a) $\text{SP}^{A,k+k_t} \leftarrow \text{Extend}(\text{SP}^{A,k}, k_t)$.
 - b) $k \leftarrow k + k_t$.
4. Reconstruct the string from $\text{SP}^{A,k}$.

Procedure *Extend* uses $\text{SP}^{A,k}$ and one round of queries in order to build $\text{SP}^{A,k+k_t}$. If at step 4 of the algorithm the value of k is $2 \log n + s$, then A will be correctly reconstructed with probability $1 - 4^{-s}$ [20]. In particular, if $s = \Omega(\log n)$ then A will be correctly reconstructed with high probability. Our goal in the next sections is to design procedure *Extend*, analyze its performance, and choose the parameters $k_0, \dots, k_T$.

The following theorem (cf. [2]) will be used to bound the number of queries.

Theorem 1 (Azuma's inequality). *Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be a function such that $|f(x) - f(x')| \leq c_i$ if x and x' differ only on the i -th coordinate. Let $Z_1, \dots, Z_n$ be independent random variables. Then,*

$$\mathbb{P}[f(Z_1, \dots, Z_n) - \mathbb{E}[f(Z_1, \dots, Z_n)] > t] \leq \exp\left(\frac{-t^2}{\sum_{i=1}^n c_i^2}\right).$$

3 Binary Spectrum

In this section, we consider the case of binary spectrum. Procedure $\text{Extend}(\text{SP}^{A,k}, \Delta)$ is as follows:

1. Let Q be the set of all strings $x_1 \cdots x_{k+\Delta}$ such that $\text{SP}^{A,k}(x_i \cdots x_{i+k-1}) = 1$ for all $i \in \{1, \dots, \Delta\}$.
2. Ask the queries in Q .

3. For every string B of length $k + \Delta$, set $\text{SP}^{A,k+\Delta}(B) = 1$ if $B \in Q$ and the answer for B was ‘yes’, and set $\text{SP}^{A,k+\Delta}(B) = 0$ otherwise.

We give a small example of procedure Extend: Let $A = \text{CGGATGAG}$, $k = 3$, and $\Delta = 2$. The set Q contains all the substrings of A of length 5 (CGGAT, GGATG, GATGA, and ATGAG). Furthermore, Q contains the string CGGAG as all its substrings of length 3 (CGG, GGA, GAG) are substrings of A , and the strings ATGAT and TGATG.

The correctness of procedure Extend is trivial. We now estimate the number of queries that are asked by the procedure. The number of queries in Q for which the answer is ‘yes’ is at most $n - (k + \Delta) + 1$. It remains to bound the number of queries for which the answer is ‘no’.

Lemma 1. *The expected number of ‘no’ queries asked by $\text{Extend}(\text{SP}^{A,k}, \Delta)$ is $O((\Delta(k + \Delta)4^{\Delta-k} + (n/4^k)\Delta^2 k 4^{\Delta-k} + (n\Delta/4^k)e^{n\Delta/4^{k-1}}) \cdot n)$.*

Proof. For each query $x_1 \cdots x_{k+\Delta}$ in Q we have $\text{SP}^{A,k}(x_1 \cdots x_k) = 1$, namely, the string $x_1 \cdots x_k$ is a substring of A . Therefore, we can estimate the number of queries in the following way: Let Y_t be the number of ‘no’ queries whose prefixes of length k are A_t^k . Then, the total number of ‘no’ queries is at most $\sum_{t=1}^{n-k+1} Y_t$. Note that for a substring that appears twice or more in A , we count the same queries several times. However, this does not significantly increase our bound on the number of queries.

In the rest of the proof, we will bound the expectation of Y_t for some fixed t . We assume that $t \leq n - (k + \Delta) + 1$ as the expectation for $t > n - (k + \Delta) + 1$ is smaller. Define the following random variables: For $s \in \{1, \dots, \Delta\}$, let Y_t^s be the number of ‘no’ queries in Q of the form $a_t \cdots a_{t+k+\Delta-s-1} b_1 \cdots b_s$, where $b_1 \neq a_{t+k+\Delta-s}$. Clearly, $Y_t = \sum_{s=1}^{\Delta} Y_t^s$. Fix some s . By definition,

$$\mathbb{E}[Y_t^s] = \sum_{b_1 \neq a_{t+k+\Delta-s}, b_2, \dots, b_s} \mathbb{P}[a_t \cdots a_{t+k+\Delta-s-1} b_1 \cdots b_s \in Q].$$

The probabilities in the sum above depend on the choice of $b_1, \dots, b_s$. Therefore, to simplify the analysis we select the letters $b_1, \dots, b_s$ at random, that is, b_1 is selected uniformly from $\Sigma - \{a_{t+k+\Delta-s}\}$, and $b_2, \dots, b_s$ are selected uniformly from Σ (Note that since $a_{t+k+\Delta-s}$ has a uniform distribution over Σ , b_1 also has a uniform distribution over Σ). Let $B = a_t \cdots a_{t+k+\Delta-s-1} b_1 \cdots b_s$, and let P_s denote the probability that $B \in Q$. We have that $\mathbb{E}[Y_t^s] = 3 \cdot 4^s \cdot P_s$.

Every k -substring of B is a substring of A , so there are indices $r_1, \dots, r_{\Delta+1}$ such that $B_i^k = A_{r_i}^k$ for $i = 1, \dots, \Delta + 1$. The sequences $A_{r_1}^k, \dots, A_{r_{\Delta+1}}^k$ will be called *supporting probes*, and a probe $A_{r_i}^k$ will be denoted by r_i . By the definition of s , $r_i = t + i - 1$ for $i = 1, \dots, \Delta - s + 1$, and $r_i \neq t + i - 1$ for $i = \Delta + 2 - s, \dots, \Delta + 1$. We need to estimate the probability that $B_i^k = A_{r_i}^k$ for $i = \Delta + 2 - s, \dots, \Delta + 1$ (we ignore the probes $r_1, \dots, r_{\Delta+1-s}$ in the rest of the proof). These equality events may not be independent: For example, suppose that $r_{\Delta-1} = r_{\Delta} = r_{\Delta+1} \leq t - k$. Then, $B_{\Delta}^k = A_{r_{\Delta}}^k$ and $B_{\Delta+1}^k = A_{r_{\Delta+1}}^k$ implies that the last $k + 1$ letters of B are identical, and it follows that

$\mathbb{P}\left[B_{\Delta-1}^k = A_{r_{\Delta-1}}^k \mid B_{\Delta}^k = A_{r_{\Delta}}^k \wedge B_{\Delta+1}^k = A_{r_{\Delta+1}}^k\right] = 1/4$. Therefore, in order to estimate the probability that $B_i^k = A_{r_i}^k$ for $i = \Delta + 2 - s, \dots, \Delta + 1$, we will consider several cases which cause these events to be dependent.

In the first case, suppose that there is a probe r_i ($i \geq \Delta + 2 - s$) that has a common letter with $a_t \cdots a_{t+k+\Delta-s}$, that is, $r_i \in I = [t - k + 1, t + k + \Delta - s]$. The event $B_i^k = A_{r_i}^k$ is composed of k equalities between the $i + j$ -th letter of B and a_{r_i+j} for $j = 0, \dots, k - 1$. Each such equality adds a requirement that either two letters of A are equal (if $i + j \leq k + \Delta - s$), or a letter in $b_1 \cdots b_s$ is equal to a letter in A . In either case, the probability that such equality happens given the previous equalities happen is exactly $1/4$, as at least one of the two letters of the equality is not restricted by the previous equalities. Therefore, for fixed i and r_i , the probability that $B_i^k = A_{r_i}^k$ is $1/4^k$. The number of ways to choose i is $s \leq \Delta$, and the number of ways to choose r_i is at most $|I| = 2k + \Delta - s \leq 2(k + \Delta)$, so the contribution of the first case to P_s is at most $2\Delta(k + \Delta)/4^k$. For the rest of the proof, assume that $r_{\Delta+2-s}, \dots, r_{\Delta+1} \notin I$.

In the second case assume that there are two probes r_i and r_j such that $|r_i - r_j| < k$ (namely, the probes have common letters) and $r_j - r_i \neq j - i$. By [3, p. 437], the probability that $B_i^k = A_{r_i}^k$ and $B_j^k = A_{r_j}^k$ is $1/4^{2k}$. The number of ways to choose i and j is $\binom{s}{2} \leq \Delta^2/2$, and the number of ways to choose r_i and r_j is at most $2kn$, so the contribution of the second case to P_s is bounded by $\Delta^2 kn/4^{2k}$.

We now consider the remaining case. We say that two probes r_i and r_j are *adjacent* if $r_j - r_i = j - i$ (in particular, every probe is adjacent to itself). For two adjacent probes r_i and r_j with $i < j$, the events $B_i^k = A_{r_i}^k$ and $B_j^k = A_{r_j}^k$ happen if and only if $B_i^{k+j-i} = A_{r_i}^{k+j-i}$. More generally, for each equivalence class of the adjacency relation, there is a corresponding equality event between a substring of A and a substring of B . Furthermore, if r_i and r_j are adjacent ($i < j$), then $B_l^k = A_{r_i+l-i}^k$ for every $l = i, \dots, j$. Therefore, we can assume w.l.o.g. that $r_l = r_i + l - i$ for $l = i, \dots, j$. Thus, each equivalence class of the adjacency relation corresponds to an interval in $\{\Delta + 2 - s, \dots, \Delta + 1\}$. More precisely, let $\Delta + 2 - s = c_1 < c_2 < \dots < c_x < c_{x+1} = \Delta + 2$ be indices such that the probes $r_{c_i}, r_{c_i+1}, \dots, r_{c_{i+1}-1}$ form an equivalence class for $i = 1, \dots, x$. We need to compute the probability that $B_{c_i}^{k-1+c_{i+1}-c_i} = A_{r_{c_i}}^{k-1+c_{i+1}-c_i}$ for $i = 1, \dots, x$. Since these events are independent (as we assumed that case 2 does not occur), the probability that all of them happen for fixed $r_{\Delta+2-s}, \dots, r_{\Delta+1}$ is

$$\prod_{i=1}^x \frac{1}{4^{k-1+c_{i+1}-c_i}} = \frac{1}{4^{\sum_{i=1}^x (k-1+c_{i+1}-c_i)}} = \frac{1}{4^{(k-1)x+s}}.$$

For fixed x , the number of ways to choose $c_1, \dots, c_x$ is $\binom{s-1}{x-1}$. After $c_1, \dots, c_x$ are chosen, the number of ways to choose $r_{\Delta+2-s}, \dots, r_{\Delta+1}$ is at most n^x . Therefore, the contribution of this case to P_s is at most

$$\sum_{x=1}^s \binom{s-1}{x-1} \frac{n^x}{4^{(k-1)x+s}} = \frac{n}{4^{k-1+s}} \sum_{x=1}^s \binom{s-1}{x-1} \left(\frac{n}{4^{k-1}}\right)^{x-1}$$

$$= \frac{n}{4^{k-1+s}} \left(1 + \frac{n}{4^{k-1}}\right)^{s-1} \leq \frac{n}{4^{k-1+s}} \cdot e^{n(s-1)/4^{k-1}}.$$

Combining the three cases, we obtain that

$$P_s \leq \frac{2\Delta(k+\Delta)}{4^k} + \frac{\Delta^2 kn}{4^{2k}} + \frac{n}{4^{k-1+s}} \cdot e^{n\Delta/4^{k-1}},$$

and

$$\mathbb{E}[Y_t] = \sum_{s=1}^{\Delta} 3 \cdot 4^s \cdot P_s \leq 4^{\Delta+1} \cdot \left(\frac{2\Delta(k+\Delta)}{4^k} + \frac{\Delta^2 kn}{4^{2k}} \right) + 3\Delta \frac{n}{4^{k-1}} e^{n\Delta/4^{k-1}}.$$

The expected number of ‘no’ queries is at most n times the last expression, so the lemma follows. $\square$

We note that we can improve the bound in Lemma 1 by reducing the bounds on the first two cases in the proof. However, this improvement does not change the bounds on the performance of our algorithms.

Lemma 2. *If $\log n \leq k \leq O(\log n)$ and $\Delta \leq 0.48 \cdot \log n$, then w.h.p., the number of ‘no’ queries asked by $\text{Extend}(SP^{A,k}, \Delta)$ is $O((n\Delta/4^k)e^{n\Delta/4^{k-1}} \cdot n) + o(n)$.*

Proof. Let Y be a random variable that counts the number of queries for which the answer is ‘no’. By Lemma 1,

$$\begin{aligned} \mathbb{E}[Y] &= O((\log^2 n + \log^3 n) \cdot 4^{-0.52 \log n} \cdot n + (n\Delta/4^k)e^{n\Delta/4^{k-1}} \cdot n) \\ &= o(n) + O((n\Delta/4^k)e^{n\Delta/4^{k-1}} \cdot n). \end{aligned}$$

The random variable Y is a function of the random variables $a_1, \dots, a_n$. A change in one letter a_i changes at most k substrings of A of length k . For a single k -substring of A , the number of strings of length $k + \Delta$ that contains it is at most $(\Delta + 1)4^\Delta = O(n^{0.48} \log n)$. Therefore, a change in one letter of A changes the number of queries by at most $O(n^{0.48} \log^2 n)$. Using Azuma’s inequality,

$$\mathbb{P}[Y - \mathbb{E}[Y] > n^{0.99}] \leq \exp\left(\frac{-n^{2 \cdot 0.99}}{n \cdot O(n^{0.48} \log^2 n)^2}\right) = e^{-\Omega(n^{0.02}/\log^4 n)}.$$

Therefore, w.h.p., the number of ‘no’ queries is $o(n) + O((n\Delta/4^k) \cdot e^{n\Delta/4^{k-1}} \cdot n)$. $\square$

Define a mapping f as follows: $f(1) = 1$ and $f(i) = 4^{f(i-1)}$ for $i > 1$. Note that $f(\log^* n) \geq \log n$. We now describe our first algorithm, called algorithm A. We use the algorithm given in Section 2, with the following parameters: $T = \max(\log^* n + 3 - c, 4)$ where c is some constant, $k_0 = \lceil \log n \rceil$, and $k_t = \min(f(t + c), \frac{1}{3} \log n)$ for $t = 1, \dots, T$.

Theorem 2. *With high probability, algorithm A reconstruct a random string of length n , and uses $O(n)$ queries in each round.*

Proof. Since $f(T+c-3) > \frac{1}{3} \log n$, we get that $\sum_{t=0}^T k_t \geq \frac{7}{3} \log n$, and therefore the algorithm reconstruct the target string with high probability.

The number of queries in the first round is $4^{k_0} \leq 4n$. Let $l_t = \sum_{i=0}^{t-1} k_i$ and $L_t = nk_t/4^{l_{t-1}}$. We claim that $L_t \leq L_1$ for all $t \geq 2$. The proof is simple as

$$L_t = \frac{nk_t}{4^{l_{t-1}}} \leq \frac{n4^{k_{t-1}}}{4^{l_{t-1}}} = \frac{n}{4^{l_{t-1}-1}} \leq L_{t-1}.$$

By Lemma 2, w.h.p., the number of queries in round t is $n + O(L_{t-1}e^{L_{t-1}} \cdot n) + o(n)$. Since $L_t \leq L_1 \leq nf(c+1)/4^{k_0-1} = O(1)$, it follows that the number of queries in each round is $O(n)$. $\square$

Algorithm B uses the following parameters: $T = 1$, $k_0 = \lceil \log n + \log \log n - \log^{(3)} n \rceil$, and $k_1 = \lceil \log n - \log \log n + 2 \log^{(3)} n \rceil$.

Theorem 3. *With probability $1 - o(1)$, the number of queries in each round of algorithm B is $O(n \log n / \log \log n)$.*

Proof. The number of queries in the first round is $4^{k_0} = O(n \log n / \log \log n)$. Let Y be the number of ‘no’ queries in the second round. By Lemma 1,

$$\begin{aligned} \mathbb{E}[Y] &= O\left(\left(\log^2 n + \frac{\log \log n}{\log n} \log^3 n\right) 4^{-2 \log \log n + 3 \log^{(3)} n} n + \log \log n \cdot e^{\log \log n} n\right) \\ &= O(\log \log n \cdot \log^{\log e} n \cdot n). \end{aligned}$$

From Markov’s inequality, with probability $1 - 1/\log^{0.1} n$, $Y \leq \mathbb{E}[Y] \cdot \log^{0.1} n = o(n \log n / \log \log n)$. $\square$

4 Trinary Spectrum

In this section, we handle the case of trinary spectrum. We use a different implementation of procedure Extend, which is based on the algorithm of [10]:

1. Let Q be the set of all strings $x_1 \cdots x_{k+j}$ such that $j \in \{1, \dots, \Delta\}$, $\text{SP}^{A,k}(x_1 \cdots x_k) \geq 1$, $\text{SP}^{A,k}(x_{j+1} \cdots x_{k+j}) \geq 1$, and $\text{SP}^{A,k}(x_i \cdots x_{i+k-1}) = 2$ for $i = 2, \dots, j$.
2. Ask the queries in Q and construct $\text{SP}^{A,k+\Delta}$.

The correctness of procedure Extend follows from [10].

Lemma 3. *If $k \geq \log n + 2$, the expected number of ‘no’ queries asked by $\text{Extend}(\text{SP}^{A,k}, \Delta)$ is $O((\Delta^2(k+\Delta)4^{\Delta-k} + \Delta^3 k(n/4^k)4^{\Delta-k} + (n/4^k)^2) \cdot n)$.*

Proof. The proof is similar to the proof of Lemma 1. We define Y_t in the same way as before. Fix some $t \leq n - (k + \Delta) + 1$. We define random variables: For $s \in \{1, \dots, \Delta\}$ and $l \in \{0, \dots, \Delta - s\}$, let $Y_t^{s,l}$ be the number of queries in Q of the form $a_t \cdots a_{t+k+l-1} b_1 \cdots b_s$, where $b_1 \neq a_{t+k+l}$. Clearly,

$$\mathbb{E}[Y_t] = \sum_{s=1}^{\Delta} \sum_{l=0}^{\Delta-s} \mathbb{E}[Y_t^{s,l}].$$

Fix some s and l , and randomly select $b_1, \dots, b_l$. Let $P_{s,l}$ be the probability that $B = a_t \cdots a_{t+k+l-1} b_1 \cdots b_s \in Q$, and we have that $E[Y_t^{s,l}] = 3 \cdot 4^s \cdot P_{s,l}$. Each k -substring of B appear at least twice in A , except the first and last ones which appear at least once. Let $r_1^1, r_2^1, r_2^2, \dots, r_{l+s}^1, r_{l+s}^2, r_{l+s+1}^1$ be indices such that $B_i^k = A_{r_i^j}^k$ for all i and j . W.l.o.g., $r_i^1 = t + i - 1$ for $i = 1, \dots, l + 1$, and $r_i^j \neq t + i - 1$ if $j = 2$ or $i \geq l + 2$. For the rest of the proof, we shall ignore the probes $r_1^1, \dots, r_{l+1}^1$ and r_{l+s+1}^1 . Our goal is to bound the probability that $B_i^k = A_{r_i^j}^k$ for $(i, j) \in \{(l + 2, 1), \dots, (l + s, 1), (2, 2), \dots, (l + s, 2)\}$. We shall denote this event by $\mathcal{E}$.

Consider the case when $r_i^j \in I = [t - k + 1, t + k + l]$ for some i and j , and the case when there are two probes r_i^j and $r_{i'}^{j'}$ for which $|r_i^j - r_{i'}^{j'}| < k$ and $r_{i'}^{j'} - r_i^j \neq i' - i$. Using the same arguments as in proof of Lemma 1, we have that the contribution of the first case to $P_{s,l}$ is at most

$$\frac{(l + s - 1) \cdot |I|}{4^k} \leq \frac{2\Delta(k + \Delta)}{4^k},$$

and the contribution of the second case to $P_{s,l}$ is at most

$$\frac{\binom{l+s}{2} \cdot 2kn}{4^{2k}} \leq \frac{\Delta^2 kn}{4^{2k}}.$$

For the rest of the proof, we assume that these cases do not occur.

Consider the equivalence classes of the adjacency relation. W.l.o.g., each equivalence class is of the form $r_i^{c_0}, r_{i+1}^{c_1}, \dots, r_{i+j}^{c_j}$. For every $i \geq l + 2$, the two indices r_i^1 and r_i^2 are interchangeable. From this fact, it follows that we can choose the indices r_i^c such that each equivalence class is of the form $r_i^c, r_{i+1}^c, \dots, r_{i+j}^c$. To prove this claim, suppose that initially $r_{l+2}^1, \dots, r_{l+s}^1, r_2^2, \dots, r_{l+s}^2$ are not assigned to a value. For $i = 2, \dots, l + 1$, we need to assign a value for r_i^2 from a set of size one, and for $i = l + 2, \dots, l + s$, we need to assign distinct values for r_i^1 and r_i^2 from a set of size two. Denote the sets of values by $R_2, \dots, R_{l+s}$. Apply the following algorithm: Let r_i^c be an unassigned probe with a minimum index i . Arbitrarily select an unused value from R_i and assign it to r_i^c . Then, for every $j > i$ and unused value $r \in R_j$ such that $r - r_i^c = j - i$, assign r to r_j^c . Repeat this process until all the values are assigned. It is easy to verify that the this algorithm generates indices with the desired property.

Now, suppose that there are x_1 equivalence classes in $r_{l+2}^1, \dots, r_{l+s}^1$, and x_2 equivalence classes in $r_2^2, \dots, r_{l+s}^2$. Then, for fixed indices, the probability that event $\mathcal{E}$ happens is $1/4^{(k-1)(x_1+x_2)+l+2s-2}$. For fixed x_1 and x_2 , the number of ways to choose the indices $r_{l+2}^1, \dots, r_{l+s}^1, r_2^2, \dots, r_{l+s}^2$ is at most $\binom{(s-1)-1}{x_1-1} \cdot \binom{(l+s-1)-1}{x_2-1} n^{x_1+x_2}$. Therefore, the contribution of this case to $P_{s,l}$ is bounded by

$$\sum_{x_1=1}^{s-1} \sum_{x_2=1}^{l+s-1} \binom{(s-1)-1}{x_1-1} \binom{(l+s-1)-1}{x_2-1} n^{x_1+x_2} \frac{1}{4^{(k-1)(x_1+x_2)+l+2s-2}}$$

$$\begin{aligned}
&= \frac{n^2}{4^{2(k-1)+l+2s-2}} \sum_{x_1=1}^{s-1} \binom{s-2}{x_1-1} \left(\frac{n}{4^{k-1}}\right)^{x_1-1} \sum_{x_2=1}^{l+s-1} \binom{l+s-2}{x_2-1} \left(\frac{n}{4^{k-1}}\right)^{x_2-1} \\
&= \frac{n^2}{4^{2k+l+2s-4}} \left(1 + \frac{n}{4^{k-1}}\right)^{l+2s-4} \leq \frac{n^2}{4^{2k+l+2s-4}} \left(1 + \frac{n}{4^{k-1}}\right)^{2(l+s)} \\
&= \frac{256}{4^s} \left(\frac{n}{4^k}\right)^2 \cdot \left(\frac{(1+n/4^{k-1})^2}{4}\right)^{l+s} \leq \frac{256}{4^s} \left(\frac{n}{4^k}\right)^2 \cdot \frac{1}{2^{l+s}}.
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathbb{E}[Y_t] &\leq \sum_{s=1}^{\Delta} \sum_{l=0}^{\Delta-s} 3 \cdot 4^s \cdot \left(\frac{2\Delta(k+\Delta)}{4^k} + \frac{\Delta^2 kn}{4^{2k}} + \frac{256}{4^s} \left(\frac{n}{4^k}\right)^2 \cdot \frac{1}{2^{l+s}} \right) \\
&\leq \Delta \cdot 4^{\Delta+1} \cdot \left(\frac{2\Delta(k+\Delta)}{4^k} + \frac{\Delta^2 kn}{4^{2k}} \right) + 768 \left(\frac{n}{4^k}\right)^2 \cdot \sum_{s=1}^{\infty} \frac{1}{2^s} \cdot \sum_{l=0}^{\infty} \frac{1}{2^l} \\
&= 8\Delta^2(k+\Delta)4^{\Delta-k} + 4\Delta^3 k(n/4^k)4^{\Delta-k} + 1536(n/4^k)^2.
\end{aligned}$$

The lemma follows by multiplying the last expression by n . $\square$

Algorithm C uses the following parameters: $T = 2$, $k_0 = \lceil \log n \rceil + 2$, $k_1 = \lceil 0.4 \log n \rceil$, and $k_2 = \lceil 0.7 \log n \rceil$.

Theorem 4. *With high probability, the number of queries in each round of algorithm C is $O(n)$.*

Proof. The number of queries in the first round is $4^{k_0} = O(n)$. Let Y and Y' be the number of ‘no’ queries in the second round and third round, respectively. By Lemma 3,

$$\mathbb{E}[Y] = O\left(\frac{\log^3 n + \log^4 n}{n^{0.6}} \cdot n + n\right) = O(n)$$

and

$$\mathbb{E}[Y'] = O\left(\frac{\log^3 n + n^{-0.4} \log^4 n}{n^{0.7}} \cdot n + n^{-0.8} \cdot n\right) = O(n^{0.4}).$$

Using Azuma’s inequality and Markov’s inequality, we obtain that w.h.p., $Y \leq \mathbb{E}[Y] + n^{0.99} = O(n)$ and $Y' \leq \mathbb{E}[Y'] \cdot n^{0.1} = o(n)$. $\square$

References

1. L. M. Adleman. Location sensitive sequencing of DNA. Technical report, University of Southern California, 1998.
2. N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley, New York, 1992.
3. R. Arratia, D. Martin, G. Reinert, and M. S. Waterman. Poisson process approximation for sequence repeats, and sequencing by hybridization. *J. of Computational Biology*, 3(3):425–463, 1996.

4. W. Bains and G. C. Smith. A novel method for nucleic acid sequence determination. *J. Theor. Biology*, 135:303–307, 1988.
5. A. Ben-Dor, I. Pe’er, R. Shamir, and R. Sharan. On the complexity of positional sequencing by hybridization. *J. Theor. Biology*, 8(4):88–100, 2001.
6. S. D. Broude, T. Sano, C. S. Smith, and C. R. Cantor. Enhanced DNA sequencing by hybridization. *Proc. Nat. Acad. Sci. USA*, 91:3072–3076, 1994.
7. R. Drmanac, I. Labat, I. Brkner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: theory of the method. *Genomics*, 4:114–128, 1989.
8. M. E. Dyer, A. M. Frieze, and S. Suen. The probability of unique solutions of sequencing by hybridization. *J. of Computational Biology*, 1:105–110, 1994.
9. A. Frieze, F. Preparata, , and E. Upfal. Optimal reconstruction of a sequence from its probes. *J. of Computational Biology*, 6:361–368, 1999.
10. A. M. Frieze and B. V. Halldórsson. Optimal sequencing by hybridization in rounds. *J. of Computational Biology*, 9(2):355–369, 2002.
11. E. Halperin, S. Halperin, T. Hartman, and R. Shamir. Handling long targets and errors in sequencing by hybridization. In *Proc. 6th Annual International Conference on Computational Molecular Biology (RECOMB ’02)*, pages 176–185, 2002.
12. S. Hannenhalli, P. A. Pevzner, H. Lewis, and S. Skiena. Positional sequencing by hybridization. *Computer Applications in the Biosciences*, 12:19–24, 1996.
13. S. A. Heath and F. P. Preparata. Enhanced sequence reconstruction with DNA microarray application. In *COCOON ’01*, pages 64–74, 2001.
14. S. A. Heath, F. P. Preparata, and J. Young. Sequencing by hybridization using direct and reverse cooperating spectra. In *Proc. 6th Annual International Conference on Computational Molecular Biology (RECOMB ’02)*, pages 186–193, 2002.
15. H. W. Leong, F. P. Preparata, W. K. Sung, and H. Willy. On the control of hybridization noise in DNA sequencing-by-hybridization. In *Proc. 2nd Workshop on Algorithms in Bioinformatics (WABI ’02)*, pages 392–403, 2002.
16. Y. Lysov, V. Floretiev, A. Khorlyn, K. Khrapko, V. Shick, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Dokl. Acad. Sci. USSR*, 303:1508–1511, 1988.
17. D. Margaritis and S. Skiena. Reconstructing strings from substrings in rounds. In *Proc. 36th Symposium on Foundation of Computer Science (FOCS 95)*, pages 613–620, 1995.
18. I. Pe’er, N. Arbili, and R. Shamir. A computational method for resequencing long dna targets by universal oligonucleotide arrays. *Proc. National Academy of Science USA*, 99:15497–15500, 2002.
19. I. Pe’er and R. Shamir. Spectrum alignment: Efficient resequencing by hybridization. In *Proc. 8th International Conference on Intelligent Systems in Molecular Biology (ISMB ’00)*, pages 260–268, 2000.
20. P. A. Pevzner, Yu. P. Lysov, K. R. Khrapko, A. V. Belyavsky, V. L. Florentiev, and A. D. Mirzabekov. Improved chips for sequencing by hybridization. *J. Biomolecular Structure and Dynamics*, 9:399–410, 1991.
21. F. Preparata and E. Upfal. Sequencing by hybridization at the information theory bound: an optimal algorithm. In *Proc. 4th Annual International Conference on Computational Molecular Biology (RECOMB ’00)*, pages 88–100, 2000.
22. R. Shamir and D. Tsur. Large scale sequencing by hybridization. *J. of Computational Biology*, 9(2):413–428, 2002.
23. S. Skiena and S. Snir. Restricting SBH ambiguity via restriction enzymes. In *Proc. 2nd Workshop on Algorithms in Bioinformatics (WABI ’02)*, pages 404–417, 2002.
24. S. Skiena and G. Sundaram. Reconstructing strings from substrings. *J. of Computational Biology*, 2:333–353, 1995.

25. S. Snir, E. Yeger-Lotem, B. Chor, and Z. Yakhini. Using restriction enzymes to improve sequencing by hybridization. Technical Report CS-2002-14, Technion, Haifa, Israel, 2002.
26. D. Tsur. Bounds for resequencing by hybridization. In *Proc. ESA '03*, to appear.

Efficient Algorithms for the Ring Loading Problem with Demand Splitting

Biing-Feng Wang, Yong-Hsian Hsieh, and Li-Pu Yeh

Department of Computer Science, National Tsing Hua University Hsinchu, Taiwan
30043, Republic of China,

bfwang@cs.nthu.edu.tw, {eric,lee}@venus.cs.nthu.edu.tw

Fax: 886-3-5723694

Abstract. Given a ring of size n and a set K of traffic demands, the ring loading problem with demand splitting (RLPW) is to determine a routing to minimize the maximum load on the edges. In the problem, a demand between two nodes can be split into two flows and then be routed along the ring in different directions. If the two flows obtained by splitting a demand are restricted to integers, this restricted version is called the ring loading problem with integer demand splitting (RLPWI). In this paper, efficient algorithms are proposed for the RLPW and the RLPWI. Both the proposed algorithms require $O(|K| + t_s)$ time, where t_s is the time for sorting $|K|$ nodes. If $|K| \geq n^\epsilon$ for some small constant $\epsilon > 0$, integer sort can be applied and thus $t_s = O(|K|)$; otherwise, $t_s = O(|K| \log |K|)$. The proposed algorithms improve the previous upper bounds from $O(n|K|)$ for both problems.

Keywords: Optical networks, rings, routing, algorithms, disjoint-set data structures

1 Introduction

Let R be a ring network of size n , in which the node-set is $\{1, 2, \dots, n\}$ and the edge-set is $E = \{(1, 2), (2, 3), \dots, (n-1, n), (n, 1)\}$. Let K be a set of traffic demands, each of which is described by an origin-destination pair of nodes together with an integer specifying the amount of traffic requirement. The ring is undirected. Each demand can be routed along the ring in any of the two directions, clockwise and counterclockwise. A demand between two nodes i and j , where $i < j$, is routed in the *clockwise* direction if it passes through the node sequence $(i, i+1, \dots, j)$, and is routed in the *counterclockwise* direction if it passes through the node sequence $(i, i-1, \dots, 1, n, n-1, \dots, j)$. The *load* of an edge is the total traffic flow passing through it. Given the ring-size n and the demand-set K , the *ring loading problem* (RLP) is to determine a routing to minimize the maximum load of the edges.

There are two kinds of RLP. If each demand in K must be routed entirely in either of the directions, the problem is called the *ring loading problem without demand splitting* (RLPWO). Otherwise, the problem is called the *ring loading*

problem with demand splitting (RLPW), in which each demand may be split between both directions. In RLPW, it is allowed to split a demand into two fractional flows. If the two flows obtained by splitting a demand are restricted to integers, this restricted version is called the *ring loading problem with integer demand splitting (RLPWI)*.

The RLP arose in the planning of optical communication networks that use bi-directional SONET (Synchronous Optical Network) rings [2,7,8]. Because of its practical significance, many researchers have turned their attention to this problem. Cosares and Saniee [2] showed that the RLPWO is NP-hard if more than one demand is allowed between the same origin-destination pair and the demands can be routed in different directions. Two approximation algorithms had been presented for the RLPWO. One was presented by Schrijver, Seymour, and Winkler [7], which has a performance guarantee of $3/2$. The other was presented by Amico, Labbe, and Maffioli [1], which has a performance guarantee of 2. For the RLPW, Schrijver, Seymour, and Winkler [7] had an $O(n^2|K|)$ -time algorithm, Vachani, Shulman, and Kubat [8] had an $O(n^3)$ -time algorithm, and Myung, Kim, and Tcha [5] had an $O(n|K|)$ -time algorithm. For the RLPWI, Lee and Chang [4] had an approximation algorithm, Schrijver, Seymour, and Winkler [7] had an pseudo-polynomial algorithm, and Vachani, Shulman, and Kubat [8] had an $O(n^3)$ -time algorithm. Very recently, Myung [6] gave some interesting properties for the RLPWI and proposed an $O(n|K|)$ -time algorithm.

In this paper, efficient algorithms are proposed for the RLPW and the RLPWI. Both the proposed algorithms require $O(|K| + t_s)$ time, where t_s is the time for sorting $|K|$ nodes. If $|K| \geq n^\epsilon$ for some small constant $\epsilon > 0$, integer sort can be applied and thus $t_s = O(|K|)$; otherwise, $t_s = O(|K| \log |K|)$. For the real world application mentioned above, $|K|$ is usually not smaller than n and thus our algorithms achieve linear time. We remark that the problem size is $|K| + 1$ instead of $|K| + n$, since a ring can be simply specified by its size n . The proposed algorithms improve the previous upper bounds from $O(n|K|)$ for both problems. They are modified versions of the algorithms in [5,6].

For easy description, throughout the remainder of this paper, we assume that $2|K| \geq n$. In case this is not true, we transform in $O(t_s)$ time the given n and K into another instance n' and K' as follows. First, we sort the distinct nodes in K into an increasing sequence S . Then, we set $n' = |S|$ and replace each node in K by its rank in S to obtain K' .

The remainder of this paper is organized as follows. In the next section, notation and preliminary results are presented. Then, in Sections 3 and 4, $O(|K| + t_s)$ -time algorithms are presented for the RLPW and the RLPWI, respectively. Finally, in Section 5, concluding remarks are given.

2 Notation and Preliminaries

Let $R = (V, E)$ be a ring of size n , where the node-set is $V = \{1, 2, \dots, n\}$ and the edge-set is $E = \{(1, 2), (2, 3), \dots, (n-1, n), (n, 1)\}$. For each i , $1 \leq i \leq n$, denote e_i as the edge $(i, (i \bmod n) + 1)$. Let K be a set of demands. For easy

description, throughout this paper, the k -th demand in K is simply denoted by k , where $1 \leq k \leq |K|$. For each $k \in K$, let $o(k)$, $d(k)$, and $r(k)$ be, respectively, the origin node, the destination node, and the amount of traffic requirement, where $o(k) < d(k)$. Assume that no two demands have the same origin-destination pair; otherwise, we simply merge them into one. For each $k \in K$, let $E_k^+ = \{e_i | o(k) \leq i \leq d(k) - 1\}$, which is the set of edges in the clockwise direction path from $o(k)$ to $d(k)$, and let $E_k^- = E \setminus E_k^+$, which is the set of edges in the counterclockwise direction path from $o(k)$ to $d(k)$. Let $\mathcal{X} = \{(x(1), x(2), \dots, x(|K|)) | x(k) \text{ is a real number and } 0 \leq x(k) \leq r(k) \text{ for each } k \in K\}$. Each $(x(1), x(2), \dots, x(|K|)) \in \mathcal{X}$ defines a *routing* for K , in which for each $k \in K$ the flow routed clockwise is $x(k)$ and the flow routed counterclockwise is $r(k) - x(k)$. Given a routing $X = (x(1), x(2), \dots, x(|K|))$, the *load* of each edge $e_i \in E$ is $g(X, e_i) = \sum_{k \in K, e_i \in E_k^+} x(k) + \sum_{k \in K, e_i \in E_k^-} (r(k) - x(k))$. The RLPW is to find a routing $X \in \mathcal{X}$ that minimizes $\max_{1 \leq i \leq n} g(X, e_i)$. The RLPWI is to find a routing $X \in \mathcal{X} \cap Z^{|K|}$ that minimizes $\max_{1 \leq i \leq n} g(X, e_i)$.

In the remainder of this section, some preliminary results are presented.

Lemma 1. *Given a routing $X = (x(1), x(2), \dots, x(|K|))$, all $g(X, e_i)$, $1 \leq i \leq n$, can be computed in $O(|K|)$ time.*

Proof. We transform the computation into the problem of computing the prefix sums of a sequence of n numbers. First, we initialize a sequence $(s(1), s(2), \dots, s(n)) = (0, 0, \dots, 0)$. Next, for each $k \in K$, we add $r(k) - x(k)$ to $s(1)$, add $-r(k) + 2x(k)$ to $s(o(k))$, and add $r(k) - 2x(k)$ to $s(d(k))$. Then, for $i = 1$ to n , we compute $g(X, e_i)$ as $s(1) + s(2) + \dots + s(i)$. It is easy to check the correctness of the above computation. The lemma holds. $\square$

Let $A = (a(1), a(2), \dots, a(n))$ be a sequence of n values. The maximum of A is denoted by $\max(A)$. The *suffix maximums* of A are elements of the sequence $(c(1), c(2), \dots, c(n))$ such that $c(i) = \max\{a(i), a(i+1), \dots, a(n)\}$. For each i , $1 \leq i \leq n$, we define the function $\pi(A, i)$ to be the largest index $j \geq i$ such that $a(j) = \max\{a(i), a(i+1), \dots, a(n)\}$. An element $a(j)$ is called a *suffix-maximum element* of A if $j = \pi(A, i)$ for some $i \leq j$. Clearly, the values of the suffix-maximum elements of A , from left to right, are strictly decreasing and the first such element is $\max(A)$. Define $\Gamma(A)$ to be the index-sequence of the suffix-maximum elements of A . Let $\Gamma(A) = (\gamma(1), \gamma(2), \dots, \gamma(q))$. According to the definitions of π and Γ , it is easy to see that $\pi(A, i) = \gamma(j)$ if and only if i is in the interval $[\gamma(j-1) + 1, \gamma(j)]$, where $1 \leq j \leq q$ and $\gamma(0) = 0$. According to the definition of suffix-maximum elements, it is not difficult to conclude the following two lemmas.

Lemma 2. *Let $S = (s(1), s(2), \dots, s(l))$ and $T = (t(1), t(2), \dots, t(m))$. Let $\Gamma(S) = (\alpha(1), \alpha(2), \dots, \alpha(g))$ and $\Gamma(T) = (\beta(1), \beta(2), \dots, \beta(h))$. Let $S \oplus T$ be the sequence $(s(1), s(2), \dots, s(l), t(1), t(2), \dots, t(m))$. If $s(\alpha(1)) \leq t(\beta(1))$, let $p = 0$; otherwise let p be the largest index such that $s(\alpha(p)) > t(\beta(1))$. Then, the sequence of suffix-maximum elements in $S \oplus T$ is $(s(\alpha(1)), s(\alpha(2)), \dots, s(\alpha(p)), t(\beta(1)), t(\beta(2)), \dots, t(\beta(h)))$.*

Lemma 3. Let $U = (u(1), u(2), \dots, u(n))$ and $\Gamma(U) = (\gamma(1), \gamma(2), \dots, \gamma(q))$. Let z be an integer, $1 \leq z \leq n$, and y be any positive number. Let g be such that $\gamma(g) = \pi(U, z)$. Let $W = (w(1), w(2), \dots, w(n))$ be a sequence such that $w(i) \leq w(\gamma(1))$ for $1 \leq i < \gamma(1)$, $w(i) = u(i) - y$ for $\gamma(1) \leq i < z$, and $w(i) = u(i) + y$ for $z \leq i \leq n$. If $w(\gamma(1)) \leq w(\gamma(g))$, let $p=0$; otherwise, let p be the largest index such that $w(\gamma(p)) > w(\gamma(g))$. Then, we have $\Gamma(W) = (\gamma(1), \gamma(2), \dots, \gamma(p), \gamma(g), \gamma(g+1), \dots, \gamma(q))$.

3 Algorithm for the RLPW

Our algorithm is a modified version of Myung, Kim, and Tcha's in [5]. Thus, we begin by reviewing their algorithm. Assume that the demands in K are pre-sorted as follows: if $o(k_1) < o(k_2)$, then $k_1 < k_2$, and if $(o(k_1) = o(k_2) \text{ and } d(k_1) > d(k_2))$, then $k_1 < k_2$. Initially, set $X = (x(1), x(2), \dots, x(|K|)) = (r(1), r(2), \dots, r(|K|))$, which indicates that at the beginning all demands are routed in the clockwise direction. Then, for each $k \in K$, the algorithm tries to reduce the maximum load by rerouting all or part of k in the counterclockwise direction. To be more precise, if $\max\{g(X, e_i) | e_i \in E_k^+\} > \max\{g(X, e_i) | e_i \in E_k^-\}$, the algorithm reroutes k until either all the demand is routed in the counterclockwise direction or the resulting X satisfies $\max\{g(X, e_i) | e_i \in E_k^+\} = \max\{g(X, e_i) | e_i \in E_k^-\}$. The algorithm is formally expressed as follows.

Algorithm 1. RLPW-1

Input: an integer n and a set K of demands

Output: a routing $X \in \mathcal{X}$ that minimizes $\max_{1 \leq i \leq n} g(X, e_i)$

begin

1. $X \leftarrow (r(1), r(2), \dots, r(|K|))$
2. $F \leftarrow (f(1), f(2), \dots, f(n))$, where $f(i) = g(X, e_i)$
3. **for** $k \leftarrow 1$ **to** $|K|$ **do**
4. **begin**
5. $m(E_k^+) \leftarrow \max\{f(i) | e_i \in E_k^+\}$
6. $m(E_k^-) \leftarrow \max\{f(i) | e_i \in E_k^-\}$
7. **if** $m(E_k^+) > m(E_k^-)$ **then** $y_k \leftarrow \min\{(m(E_k^+) - m(E_k^-))/2, r(k)\}$
8. **else** $y_k \leftarrow 0$
9. $x(k) \leftarrow r(k) - y_k$ /* Reroute y_k units in counterclockwise direction. */
10. Update F by adding y_k to each $f(i)$ with $e_i \in E_k^-$ and subtracting y_k from each $f(i)$ with $e_i \in E_k^+$
11. **end**
12. **return** (X)

end

The bottleneck of Algorithm 1 is the computation of $m(E_k^+)$ and $m(E_k^-)$ for each $k \in K$. In order to obtain a linear time solution, some properties of Algorithm 1 are discussed in the following. Let $X_0 = (r(1), r(2), \dots, r(|K|))$

and X_k be the X obtained after the rerouting step is performed for $k \in K$. For $0 \leq k \leq |K|$, let $F_k = (f_k(1), f_k(2), \dots, f_k(n))$, where $f_k(i) = g(X_k, e_i)$. According to the execution of Algorithm 1, once an edge becomes a maximum load edge at some iteration, it remains as such in the remaining iterations. Let $M_k = \{e_i | f_k(i) = \max(F_k), 1 \leq i \leq n\}$, which is the set of the maximum load edges with respect to X_k . We have the following.

Lemma 4. [5] *For each $k \in K$, $M_{k-1} \subseteq M_k$.*

Since $m(E_k^+) > m(E_k^-)$ if and only if $m(E_k^+) = \max(F_{k-1})$ and $m(E_k^-) \neq \max(F_{k-1})$, we have the following lemma.

Lemma 5. [5] *For each $k \in K$, $y_k > 0$ if and only if $E_k^+ \supseteq M_{k-1}$.*

Consider the computation of $m(E_k^+)$ in Algorithm 1. If $m(E_k^+) = \max(F_{k-1})$, y_k is computed as $\min\{(\max(F_{k-1}) - m(E_k^-))/2, r(k)\}$. Assume that $m(E_k^+) \neq \max(F_{k-1})$. In this case, we must have $m(E_k^-) = \max(F_{k-1})$. Thus, $m(E_k^+) < m(E_k^-)$ and y_k should be computed as 0, which is irrelevant to the value of $m(E_k^+)$. Since $m(E_k^-) = \max(F_{k-1})$, in this case, we can also compute y_k as $\min\{(\max(F_{k-1}) - m(E_k^+))/2, r(k)\}$. Therefore, to determine y_k , it is not necessary for us to compute $m(E_k^+)$. What we need is the value of $\max(F_{k-1})$. The value of $\max(F_0)$ can be computed in $O(n)$ time. According to Lemma 4 and Line 10 of Algorithm 1, after y_k has been determined we can compute $\max(F_k)$ as $\max(F_{k-1}) - y_k$.

Next, consider the computation of $m(E_k^-)$. In order to compute all $m(E_k^-)$ efficiently, we partition E_k^- into two subsets A_k and B_k , where $A_k = \{e_i | 1 \leq i < o(k)\}$ and $B_k = \{e_i | d(k) \leq i \leq n\}$. For each $k \in K$, we define $m(A_k) = \max\{f_{k-1}(i) | e_i \in A_k\}$ and $m(B_k) = \max\{f_{k-1}(i) | e_i \in B_k\}$. Then, $m(E_k^-) = \max\{m(A_k), m(B_k)\}$. We have the following.

Lemma 6. *Let $k \in K$. If there is an iteration $i < k$ such that $y_i > 0$ and $o(k) > d(i)$, then $y_j = 0$ for all $j \geq k$.*

Proof. Assume that there exists a such i . Since $y_i > 0$, by Lemma 5 we have $E_i^+ \supseteq M_{i-1}$. Consider a fixed $j \geq k$. Since $o(j) \geq o(k) > d(i)$, E_j^+ cannot include M_{i-1} . Furthermore, since by Lemma 4 $M_{j-1} \supseteq M_{i-1}$, E_j^+ cannot include M_{j-1} . Consequently, by Lemma 5, we have $y_j = 0$. Therefore, the lemma holds. $\square$

According to Lemma 6, we may maintain in Algorithm 1 a variable d_{min} to record the current smallest $d(i)$ with $y_i > 0$. Then, at each iteration $k \in K$, we check whether $o(k) > d_{min}$ and once the condition is true, we skip the rerouting for all $j \geq k$. Based upon the above discussion, we present a modified version of Algorithm 1 as follows.

Algorithm 2. RLPW-2

Input: an integer n and a set K of demands

Output: a routing $X \in \mathcal{X}$ that minimizes $\max_{1 \leq i \leq n} g(X, e_i)$

begin

1. $X \leftarrow (r(1), r(2), \dots, r(|K|))$

```

2.  $F_0 \leftarrow (f_0(1), f_0(2), \dots, f_0(n))$ , where  $f_0(i) = g(X, e_i)$ 
3.  $\max(F_0) \leftarrow \max\{f_0(i) | e_i \in E\}$ 
4.  $d_{\min} \leftarrow \infty$ 
5. for  $k \leftarrow 1$  to  $|K|$  do
6. begin
7.   if  $o(k) > d_{\min}$  then return  $(X)$ 
8.    $m(A_k) \leftarrow \max\{f_{k-1}(i) | e_i \in A_k\}$ 
9.    $m(B_k) \leftarrow \max\{f_{k-1}(i) | e_i \in B_k\}$ 
10.   $y_k \leftarrow \min\{(\max(F_{k-1}) - m(A_k))/2, (\max(F_{k-1}) - m(B_k))/2, r(k)\}$ 
11.   $x(k) \leftarrow r(k) - y_k$ 
12.   $\max(F_k) \leftarrow \max(F_{k-1}) - y_k$ 
13.  if  $y_k > 0$  and  $d(k) < d_{\min}$  then  $d_{\min} \leftarrow d(k)$ 
14. end
15. return  $(X)$ 
end

```

In the remainder of this section, we show that Algorithm 2 can be implemented in linear time. The values of $m(A_k)$, $m(B_k)$, and y_k are defined on the values of F_{k-1} . In Line 2, we compute F_0 in $O(|K|)$ time. Before presenting the details, we remark that our implementation does not compute the whole sequences of all F_{k-1} . Instead, we maintain only their information that is necessary for determining $m(A_k)$, $m(B_k)$, and y_k .

First, we describe the determination of $m(A_k)$, which is mainly based upon the following two lemmas.

Lemma 7. *For each $k \in K$, if Algorithm 2 does not terminate at Line 7, then $f_{k-1}(i) = f_0(i) - \sum_{1 \leq i \leq k-1} y_i$ for $o(k-1) \leq i < o(k)$.*

Proof. Let e_i be an edge such that $o(k-1) \leq i < o(k)$. We prove this lemma by showing that $e_i \in E_j^+$ for all $j \leq k-1$ and $y_j > 0$. Consider a fixed $j \leq k-1$ with $y_j > 0$. Since $o(j) \leq o(k-1) \leq i$, $o(j)$ is on the left side of e_i . Since Algorithm 2 does not terminate at Line 7, we have $i < o(k) \leq d_{\min} \leq d(j)$. Thus, $d(j)$ is on the right side of e_i . Therefore, $e_i \in E_j^+$ and the lemma holds. $\square$

Lemma 8. *For each $k \in K$, if Algorithm 2 does not terminate at Line 7, then $m(A_k) = \max\{m(A_{k-1}) + y_{k-1}, \max\{f_{k-1}(i) | o(k-1) \leq i < o(k)\}\}$, where $m(A_0) = 0$, $y_0 = 0$, and $o(0) = 1$.*

Proof. Recall that $m(A_k) = \max\{f_{k-1}(i) | 1 \leq i < o(k)\}$. For $k = 1$, since $m(A_0) = 0$, $y_0 = 0$, and $o(0) = 1$, the lemma holds trivially. Assume that $k \geq 2$. In the following, we complete the proof by showing that $m(A_{k-1}) + y_{k-1} = \max\{f_{k-1}(i) | 1 \leq i < o(k-1)\}$. By induction, $m(A_{k-1}) = \max\{f_{k-2}(i) | 1 \leq i < o(k-1)\}$. According to Line 10 of Algorithm 1, we have $f_{k-1}(i) = f_{k-2}(i) + y_{k-1}$ for $1 \leq i < o(k-1)$. Thus, $\max\{f_{k-1}(i) | 1 \leq i < o(k-1)\} = \max\{f_{k-2}(i) + y_{k-1} | 1 \leq i < o(k-1)\} = m(A_{k-1}) + y_{k-1}$. Therefore, the lemma holds. $\square$

According to Lemmas 7 and 8, we compute each $m(A_k)$ as follows. During the execution of Algorithm 2, we maintain an additional variable y^* such that at the beginning of each iteration k , the value of y^* is $\sum_{1 \leq i \leq k-1} y_i$. Then, for each $k \in K$, if Algorithm 2 does not terminate at Line 7, we compute $m(A_k) = \max\{m(A_{k-1}) + y_{k-1}, \max\{f_0(i) - y^* \mid o(k-1) \leq i < o(k)\}\}$ in $O(o(k) - o(k-1))$ time by using $m(A_{k-1})$, y_{k-1} , F_0 , and y^* . Since $2|K| \geq n$ and the origins $o(k)$ are non-decreasing integers between 1 and n , the computation for all $m(A_k)$ takes $O(|K|)$ time.

Next, we describe the determination of $m(B_k)$ and y_k , which is the most complicated part of our algorithm. By definition, $m(B_k) = \max\{f_{k-1}(i) \mid d(k) \leq i \leq n\} = f_{k-1}(\pi(F_{k-1}, d(k)))$. Thus, maintaining the function π for F_{k-1} is useful for computing $m(B_k)$. Let $\Gamma(F_{k-1}) = (\gamma(1), \gamma(2), \dots, \gamma(q))$. For each j , $1 \leq j \leq q$, we have $\pi(F_{k-1}, i) = \gamma(j)$ for every i in the interval $[\gamma(j-1) + 1, \gamma(j)]$, where $\gamma(0) = 0$. Thus, we call $[\gamma(j-1) + 1, \gamma(j)]$ the *domain-interval* of $\gamma(j)$. Let U_{k-1} be the sequence of domain-intervals of the elements in $\Gamma(F_{k-1})$. The following lemma, which can be obtained from Lemma 3, shows that U_k can be obtained from U_{k-1} by simply merging the domain-intervals of some consecutive elements in $\Gamma(F_{k-1})$.

Lemma 9. *Let $\Gamma(F_{k-1}) = (\gamma(1), \gamma(2), \dots, \gamma(q))$. Let g be such that $\gamma(g) = \pi(F_{k-1}, d(k))$. If $f_{k-1}(\gamma(1)) - y_k = f_{k-1}(\gamma(g)) + y_k$, let $p=0$; otherwise, let p be the largest index such that $f_{k-1}(\gamma(p)) - y_k > f_{k-1}(\gamma(g)) + y_k$. Then, we have $\Gamma(F_k) = (\gamma(1), \gamma(2), \dots, \gamma(p), \gamma(g), \gamma(g+1), \dots, \gamma(q))$.*

Based upon Lemma 9, we maintain U_{k-1} by using an interval union-find data structure, which is defined as follows. Let I_n be the interval $[1, n]$. Two intervals in I_n are *adjacent* if they can be obtained by splitting an interval. A *partition* of I_n is a sequence of disjoint intervals whose union is I_n . An *interval union-find data structure* is one that initially represents some partition of I_n and supports a sequence of two operations: $\text{FIND}(i)$, which returns the representative of the interval containing i , and $\text{UNION}(i, j)$, which unites the two adjacent intervals containing i and j , respectively, into one. The representative of an interval may be any integer contained in it. Gabow and Tarjan had the following result.

Lemma 10. *[3] A sequence of m FIND and at most $n - 1$ UNION operations on any partition of I_n can be done in $O(n + m)$ time.*

Let $\Gamma(F_{k-1}) = (\gamma(1), \gamma(2), \dots, \gamma(q))$. For convenience, we let each $\gamma(i)$ be the representative of its domain-interval such that $\pi(F_{k-1}, d(k))$ can be determined by simply performing $\text{FIND}(d(k))$. By Lemma 9, U_k can be obtained from U_{k-1} by performing a sequence of UNION operations. In order to obtain U_k in such a way, we need the representatives $\gamma(p+1)$, $\dots$, and $\gamma(g-1)$. Therefore, we maintain an additional linked list L_{k-1} to chain all representatives in U_{k-1} together such that for any given $\gamma(i)$, we can find $\gamma(i-1)$ in $O(1)$ time.

Now, we can determine $\pi(F_{k-1}, d(k))$ efficiently. However, since $m(B_k) = f_{k-1}(\pi(F_{k-1}, d(k)))$, what we really need is the value of $f_{k-1}(\pi(F_{k-1}, d(k)))$. At this writing, the author is not aware of any efficient way to compute the values

for all $k \in K$. Fortunately, in some case, it is not necessary to compute the value. Since $y_k = \min\{(max(F_{k-1}) - m(A_k))/2, (max(F_{k-1}) - m(B_k))/2, r(k)\}$, the value is needed only when $max(F_{k-1}) - m(B_k) < \min\{max(F_{k-1}) - m(A_k), 2r(k)\}$. Therefore, we maintain further information about F_{k-1} such that whether $max(F_{k-1}) - m(B_k) < \min\{max(F_{k-1}) - m(A_k), 2r(k)\}$ can be determined and in case it is true, the value of $m(B_k)$ can be computed. Let $\Gamma(F_{k-1}) = (\gamma(1), \gamma(2), \dots, \gamma(q))$. We associate with each representative $\gamma(i)$ in L_{k-1} a value $\delta(i)$, where $\delta(i)$ is 0 if $i=1$ and otherwise $\delta(i)$ is the difference $f_{k-1}(\gamma(i-1)) - f_{k-1}(\gamma(i))$. Define $\Delta(F_{k-1})$ to be the sequence $(\delta(1), \delta(2), \dots, \delta(q))$. Clearly, for any $i < j$, the difference between $f_{k-1}(\gamma(i))$ and $f_{k-1}(\gamma(j))$ is $\sum_{i+1 \leq z \leq j} \delta(z)$. And, since $f_{k-1}(\gamma(1)) = max(F_{k-1})$, the difference between $max(F_{k-1})$ and $f_{k-1}(\gamma(i))$ is $\sum_{2 \leq z \leq i} \delta(z)$. The maintainance of $\Delta(F_{k-1})$ can be done easily by using the following lemma.

Lemma 11. *Let $\Gamma(F_{k-1}) = (\gamma(1), \gamma(2), \dots, \gamma(q))$ and $\Delta(F_{k-1}) = (\delta(1), \delta(2), \dots, \delta(q))$. Let g and p be defined as in Lemma 9. Then, we have $\Delta(F_k) = (\delta(1), \delta(2), \dots, \delta(p), \delta', \delta(g+1), \delta(g+2), \dots, \delta(q))$, where $\delta' = \delta(p+1) + \delta(p+2) + \dots + \delta(g) - 2y_k$.*

Proof. By Lemma 9, we have $\Gamma(F_k) = (\gamma(1), \gamma(2), \dots, \gamma(p), \gamma(g), \gamma(g+1), \dots, \gamma(q))$. Since $f_k(\gamma(p)) - f_k(\gamma(g)) = (f_{k-1}(\gamma(p)) - y_k) - (f_{k-1}(\gamma(g)) + y_k)$, we have $f_k(\gamma(p)) - f_k(\gamma(g)) = \delta(p+1) + \delta(p+2) + \dots + \delta(g) - 2y_k$. Clearly, we have $f_k(\gamma(i-1)) - f_k(\gamma(i)) = f_{k-1}(\gamma(i-1)) - f_{k-1}(\gamma(i))$ for both $2 \leq i \leq p$ and $g < i \leq q$. By combining these two statements, we obtain $\Delta(F_k) = (\delta(1), \delta(2), \dots, \delta(p), \delta(p+1) + \delta(p+2) + \dots + \delta(g) - 2y_k, \delta(g+1), \delta(g+2), \dots, \delta(q))$. Thus, the lemma holds. $\square$

Now, we are ready to describe the detailed computation of y_k , which is done by using $m(A_k)$, $max(F_{k-1})$, $r(k)$, U_{k-1} , L_{k-1} , and $\Delta(F_{k-1})$. First, we perform $\text{FIND}(d(k))$ to get $\gamma(g) = \pi(F_{k-1}, d(k))$. Next, by traveling along the list L_{k-1} , starting at $\gamma(g)$, we compute the largest p such that $\delta(p+1) + \delta(p+2) + \dots + \delta(g) > \min\{max(F_{k-1}) - m(A_k), 2r(k)\}$. In case $\delta(1) + \delta(2) + \dots + \delta(g) \leq \min\{max(F_{k-1}) - m(A_k), 2r(k)\}$, p is computed as 0. Then, if $p > 0$, we conclude that $max(F_{k-1}) - m(B_k) = max(F_{k-1}) - f_{k-1}(\gamma(g)) > \min\{max(F_{k-1}) - m(A_k), 2r(k)\}$ and thus y_k is computed as $\min\{(max(F_{k-1}) - m(A_k))/2, r(k)\}$; otherwise, we compute $m(B_k) = f_{k-1}(\gamma(g)) = max(F_{k-1}) - (\delta(1) + \delta(2) + \dots + \delta(g))$ and then compute y_k as $(max(F_{k-1}) - m(B_k))/2$. Since $g - p - 1 = |\Gamma(F_{k-1})| - |\Gamma(F_k)|$, the above computation takes $t_f + O(|\Gamma(F_{k-1})| - |\Gamma(F_k)|)$ time, where t_f is the time for performing a FIND operation. After y_k is computed, we obtain U_k , L_k , and $\Delta(F_k)$ from U_{k-1} , L_{k-1} , and $\Delta(F_{k-1})$ in $O(|\Gamma(F_{k-1})| - |\Gamma(F_k)|) + (|\Gamma(F_{k-1})| - |\Gamma(F_k)|) \times t_u$ time according to Lemmas 9 and 11, where t_u is the time for performing a UNION operation.

Theorem 1. *The RLPW can be solved in $O(|K| + t_s)$ time, where t_s is the time for sorting $|K|$ nodes.*

Proof. We prove this theorem by showing that Algorithm 2 can be implemented in $O(|K|)$ time. Note that we had assumed $2|K| \geq n$. The time for compute X ,

F_0 , $\max(F_0)$, and $d_{\min}$ in Lines 1~4 is $O(|K|)$. Before starting the rerouting, we set $y^* = 0$, $m(A_0) = 0$, $y_0 = 0$, and initialize U_0 , L_0 , and $\Delta(F_0)$ in $O(n)$ time. Consider the rerouting in Lines 7~13 for a fixed $k \in K$. In Line 8, by using Lemmas 7 and 8, we compute $m(A_k)$ by using $m(A_{k-1})$, y_{k-1} , F_0 and y^* in $O(o(k) - o(k-1))$ time. In Lines 9 and 10, we compute y_k in $t_f + O(|\Gamma(F_{k-1})| - |\Gamma(F_k)|)$ time by using $m(A_k)$, $\max(F_{k-1})$, $r(k)$, U_{k-1} , L_{k-1} , and $\Delta(F_{k-1})$. Lines 7, 11, 12, and 13 take $O(1)$ time. Before starting the next iteration, we add y_k to y^* and obtain U_k , L_k , and $\Delta(F_k)$ from U_{k-1} , L_{k-1} , and $\Delta(F_{k-1})$ in $O(|\Gamma(F_{k-1})| - |\Gamma(F_k)|) + (|\Gamma(F_{k-1})| - |\Gamma(F_k)|) \times t_u$ time. In total, the rerouting time for a fixed $k \in K$ is $t_f + (|\Gamma(F_{k-1})| - |\Gamma(F_k)|) \times t_u + O(o(k) - o(k-1) + |\Gamma(F_{k-1})| - |\Gamma(F_k)|)$.

Since the origins $o(k)$ are non-decreasing and the sizes of $\Gamma(F_{k-1})$ are non-increasing, $\sum_{1 \leq i \leq |K|} O(o(k) - o(k-1) + |\Gamma(F_{k-1})| - |\Gamma(F_k)|) = O(|K|)$. At most $|K|$ FIND and $n-1$ UNION operations may be performed. Therefore, the overall time complexity of Algorithm 2 is $O(|K| + |K| \times t_f + n \times t_u)$, which is $O(|K|)$ by applying Gabow and Tarjan's result in Lemma 10. Consequently, the theorem holds. $\square$

4 Algorithm for the RLPWI

The algorithm proposed by Myung for the RLPWI in [7] consists of two phases. In the first phase, an optimal solution X for the RLPW is found. Then, if $X \notin \mathcal{X} \cap \mathcal{Z}^{|K|}$, the second phase is performed, in which demands are rerouted until all $x(k)$ become integers. The bottleneck of Myung's algorithm is the computation of X in the first phase and the computation of all $g(X, e_i)$ in the second phase. By using Theorem 1 and Lemma 1, it is easy to implement Myung's algorithm in $O(|K| + t_s)$ time.

Theorem 2. *The RLPWI can be solved in $O(|K| + t_s)$ time, where t_s is the time for sorting $|K|$ nodes.*

5 Concluding Remarks

In this paper, an $O(|K| + t_s)$ -time algorithm was firstly proposed for the RLPW. Then, by applying it to Myung's algorithm in [6], the RLPWI was solved in the same time. The proposed algorithms take linear time when $|K| \geq n^\epsilon$ for some small constant $\epsilon > 0$. They improved the previous upper bounds from $O(n|K|)$ for both problems.

Myung, Kim, and Tcha's algorithm for the RLPW in [5] motivated studies on the following interesting data structure. Let $X = (x_1, x_2, \dots, x_n)$ be a sequence of n values. A *range increase-decrease-maximum data structure* is one that initially represents X and supports a sequence of three operations: INCREASE(i, j, y), which adds y to every element in $(x_i, x_{i+1}, \dots, x_j)$, DECREASE(i, j, y), which subtracts y from every element in $(x_i, x_{i+1}, \dots, x_j)$, and MAXIMUM(i, j),

which returns the maximum in $(x_i, x_{i+1}, \dots, x_j)$. By using the well-known segment trees, it is not difficult to implement a data structure to support each of the three operations in $O(\log n)$ time. To design a more efficient implementation of such data structure is also worth of further study.

References

1. M. D. Amico, M. Labbe, and F. Maffioli, "Exact solution of the SONET ring loading problem," *Operations Research Letters*, vol. 25, pp. 119–129, 1999.
2. S. Cosares and I. Saniee, "An optimal problem related to balancing loads on SONET rings," *Telecommunication Systems*, vol. 3, pp. 165–181, 1994.
3. H. N. Gabow and R. E. Tarjan, "A linear-time algorithm for a special case of disjoint set union," *Journal of Computer and System Sciences*, vol. 30, pp. 209–221, 1985.
4. C. Y. Lee and S. G. Chang, "Balancing loads on SONET rings with integer demand splitting," *Computers Operations Research*, vol. 24, pp. 221–229, 1997.
5. Y.-S. Myung, H.-G. Kim, and D.-W. Tcha, "Optimal load balancing on SONET bidirectional rings," *Operations Research*, vol. 45, pp. 148–152, 1997.
6. Y.-S. Myung, "An efficient algorithm for the ring loading problem with integer demand splitting," *SIAM Journal on Discrete Mathematics*, vol. 14, no. 3, pp. 291–298, 2001.
7. A. Schrijver, P. Seymour, and P. Winkler, "The ring loading problem," *SIAM Journal on Discrete Mathematics*, vol. 11, pp. 1–14, 1998.
8. R. Vachani, A. Shulman, and P. Kubat, "Multi-commodity flows in ring networks," *INFORMS Journal on Computing*, vol. 8, pp. 235–242, 1996.

Seventeen Lines and One-Hundred-and-One Points

Gerhard J. Woeginger

University of Twente, The Netherlands
g.j.woeginger@math.utwente.nl

Abstract. We investigate a curious problem from additive number theory: Given two positive integers S and Q , does there exist a sequence of positive integers that add up to S and whose squares add up to Q ? We show that this problem can be solved in time polynomially bounded in the logarithms of S and Q .

As a consequence, also the following question can be answered in polynomial time: For given numbers n and m , do there exist n lines in the Euclidean plane with exactly m points of intersection?

1 Introduction

John Herivel relates the following story in his biography [2, p.244] of the mathematical physicist Joseph Fourier (1768–1830). In 1788, Fourier corresponded with his friend and teacher C.L. Bonard, a professor of mathematics at Auxerre. In one of his letters, Fourier sent the following teaser: “*Here is a little problem of rather singular nature. It occurred to me in connection with certain propositions in Euclid we discussed on several occasions. Arrange 17 lines in the same plane so that they give 101 points of intersection. It is to be assumed that the lines extend to infinity, and that no point of intersection belongs to more than two lines.*” Fourier suggested to analyze this problem by considering the ‘general’ case. One solution to Fourier’s problem is to use four families of parallel lines with 2, 3, 4, and 8 lines, respectively. This yields a total number of $2 \times 3 + 2 \times 4 + 2 \times 8 + 3 \times 4 + 3 \times 8 + 4 \times 8 = 101$ intersection points. A closer analysis of this problem (see for instance Turner [3]) reveals that there are three additional solutions that use (a) four families with 1, 5, 5, 6 lines, (b) five families with 1, 2, 3, 3, 8 lines, and (c) six families with 1, 1, 1, 2, 4, 8 lines.

The ‘general’ case of Fourier’s problem would probably be to decide for given numbers n and m whether there exist n lines in the Euclidean plane that give exactly m points of intersection. If two lines are parallel, they do not intersect; if two lines are non-parallel, then they contribute exactly one intersection point. Let us assume that there are k families of parallel lines, where the i -th family ($i = 1, \dots, k$) consists of n_i lines. Then every line in the i -th family intersects the $n - n_i$ lines in all the other families. Since in this argument every intersection point is counted twice, we get that $\sum_{i=1}^k n_i (n - n_i) = 2m$. Together with $\sum_{i=1}^k n_i = n$ this condition simplifies to $\sum_{i=1}^k n_i^2 = n^2 - 2m$. Hence, we have arrived at a special case of the following problem.

Problem 1 (*Fourier's general problem*)

For an input consisting of two positive integers S and Q , decide whether there exist positive integers $x_1, \dots, x_k$ with $\sum_{i=1}^k x_i = S$ and $\sum_{i=1}^k x_i^2 = Q$.

Note that the instance size in this problem is $\log S + \log Q$, the number of bits to write down S and Q . Fourier's general problem is straightforward to solve by dynamic programming within a time complexity that is polynomial in S and Q , but: this time complexity would be exponential in the input size.

Let us start with investigating the case $S = 10$. Ten minutes of scribbling on a piece of paper yield that for $S = 10$ the following values of the square sum Q yield YES-instances for Fourier's general problem:

10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
42, 44, 46, 50, 52, 54, 58, 66, 68, 82, 100.

A first (not surprising) observation is that the extremal values on this list are $Q = 10$ and $Q = 100$. By the super-additivity $(x + y)^2 \geq x^2 + y^2$ of the square function, the smallest feasible value of Q equals S (in which case $k = S$, and $x_i \equiv 1$ for $i = 1, \dots, S$) and the largest feasible value of Q equals S^2 (in which case $k = 1$ and $x_1 = S$). Another thing that catches one's eye is that all the listed values are even. But again that's not surprising at all. An integer x always has the same parity as its square x^2 , and thus also the two integers $S = \sum_{i=1}^k x_i$ and $Q = \sum_{i=1}^k x_i^2$ must have the same parity. Hence, the even value $S = 10$ enforces an even value of Q .

A more interesting property of this list is that it contains *all* the even numbers from $Q = 10$ up to $Q = 46$; then there is a gap around 48, then another gap around 56, and afterwards the numbers become quite chaotic. Where does the highly regular structure in the first half of the list come from? Out of pure accident? No, Lemma 3 in Section 2 proves that all such lists for all values of S will show similar regularities at their beginning. And are we somehow able to control the chaotic behavior in the second half of these lists? Yes, we are: Lemma 4 in Section 2 explains the origins of this chaotic behavior. This lemma allows us to (almost) guess one of the integers x_i in the representation $Q = \sum_{i=1}^k x_i^2$, and thus to reduce the instance to a smaller one.

Based on these ideas, this short note will design a polynomial time algorithm for Fourier's general problem. Section 2 derives some important properties of the problem, and Section 3 turns these properties into an algorithm.

2 Structural Results

We call a pair (S, Q) of positive integers *admissible*, if there exists a k -tuple $(x_1, \dots, x_k)$ of positive integers with $\sum_{i=1}^k x_i = S$ and $\sum_{i=1}^k x_i^2 = Q$. In this case, the k -tuple $(x_1, \dots, x_k)$ is called a *certificate* for (S, Q) .

Observation 2 Assume that (S, Q) is an admissible pair. Then S and Q are of the same parity, $S \leq Q \leq S^2$ holds, and for any positive integer y the pair $(S + y, Q + y^2)$ is also admissible. $\square$

Lemma 3 *Let S and Q be two positive integers that are of the same parity and that satisfy the inequalities*

$$S \leq Q \leq S(S - 6\sqrt{S}). \quad (1)$$

Then the pair (S, Q) is admissible.

Proof. For ease of exposition, we introduce a function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ by $f(z) = z(z - 6\sqrt{z})$. The proof is done by induction on S . A (straightforward) computer search verifies that the statement in the theorem holds true for all $S \leq 1603$. In the inductive step, we consider two integers $S \geq 1604$ and Q of the same parity that satisfy the inequalities in (1), that is, $S \leq Q \leq f(S)$. We will show that either for $x = 1$ or for $x = \lfloor S - 3\sqrt{S} - 6 \rfloor$ the pair $(S - x, Q - x^2)$ is an admissible pair. This will complete the proof.

If $Q - 1 \leq f(S - 1)$ holds, then the pair $(S - 1, Q - 1)$ is admissible by the inductive hypothesis, and we are done. Hence, we will assume from now on that

$$Q \geq f(S - 1) = (S - 1)^2 - 6(S - 1)^{3/2} > S^2 - 6S^{3/2} - 2S. \quad (2)$$

In other words, Q is sandwiched between $f(S) - 2S$ and $f(S)$. We define $x = \lfloor S - 3\sqrt{S} - 6 \rfloor$. Furthermore, we define a real α via the equation $x = S - 3\sqrt{S} - \alpha$; note that $6 \leq \alpha < 7$. With this we get that

$$\begin{aligned} Q - x^2 &> (S^2 - 6S^{3/2} - 2S) - (S^2 - 6S^{3/2} + 9S + \alpha^2 - 2\alpha S + 6\alpha\sqrt{S}) \\ &= \alpha(2S - 6\sqrt{S}) - 11S - \alpha^2 \geq 3\sqrt{S} + \alpha = S - x. \end{aligned} \quad (3)$$

Here we first used (2), and then that $6 \leq \alpha < 7$ and $S \geq 1604$. By similar arguments we derive by using (1) that

$$\begin{aligned} Q - x^2 &\leq \alpha(2S - 6\sqrt{S}) - 9S - \alpha^2 \\ &\leq (9S + \alpha^2 + 6\alpha\sqrt{S}) - 6(3\sqrt{S} + \alpha)^{3/2} = f(S - x). \end{aligned} \quad (4)$$

Summarizing, (3) and (4) yield that $S - x \leq Q - x^2 \leq f(S - x)$. Therefore, the pair $(S - x, Q - x^2)$ is admissible by the inductive hypothesis, and the argument is complete. $\square$

Lemma 4 *Let (S, Q) be an admissible pair that satisfies $S(S - 6\sqrt{S}) < Q \leq S^2$. Furthermore, let $(x_1, \dots, x_k)$ be a certificate for (S, Q) with $\sum_{i=1}^k x_i = S$ and $\sum_{i=1}^k x_i^2 = Q$, and let $\xi := \max_{1 \leq i \leq k} x_i$. Then ξ satisfies*

$$\frac{1}{2}(S + \sqrt{2Q - S^2}) \leq \xi \leq \sqrt{Q}. \quad (5)$$

If $S \geq 8061$, then there are at most five values that ξ can possibly take, and all these values are greater or equal to $S - 4\sqrt{S}$.

Proof. The upper bound in (5) follows since $\xi^2 \leq Q$. For the lower bound, suppose first for the sake of contradiction that $\xi \leq S/2$. Then $Q = \sum_{i=1}^k x_i^2 \leq 2(S/2)^2 = S^2/2$. But the conditions in the lemma yield that $S^2/2 < S(S - 6\sqrt{S}) \leq Q$, a clear contradiction. Therefore $\frac{1}{2}S < \xi$. Next, since $(S - \xi, Q - \xi^2)$ is an admissible pair, we derive from Observation 2 that ξ must satisfy the inequality $Q - \xi^2 \leq (S - \xi)^2$. The two roots of the underlying quadratic equation are $\xi_1 = \frac{1}{2}(S - \sqrt{2Q - S^2})$ and $\xi_2 = \frac{1}{2}(S + \sqrt{2Q - S^2})$, and the inequality is satisfied if and only if $\xi \leq \xi_1$ or $\xi \geq \xi_2$. Since $\xi \leq \xi_1$ would violate $\frac{1}{2}S < \xi$, we conclude that $\xi \geq \xi_2$ must hold true. This yields the lower bound on ξ as claimed in (5).

Next, we will estimate the distance between the upper and the lower bound in (5) for the case where $S \geq 8061$. Let us fix S for the moment, and let us consider the difference

$$\Delta_S(Q) := \sqrt{Q} - \frac{1}{2}(S + \sqrt{2Q - S^2})$$

between these two bounds in terms of Q . The value Q ranges from $S(S - 6\sqrt{S})$ to S^2 . The first derivative of $\Delta_S(Q)$ equals $\frac{1}{2}(1/\sqrt{Q} - 1/\sqrt{2Q - S^2}) < 0$, and therefore the function $\Delta_S(Q)$ is strictly decreasing for $S(S - 6\sqrt{S}) < Q \leq S^2$. Hence, for fixed S the difference $\Delta_S(Q)$ is maximized at $Q = S(S - 6\sqrt{S})$ where it takes the value

$$\Delta^*(S) := \sqrt{S^2 - 6S^{3/2}} - \frac{1}{2}(S + \sqrt{S^2 - 12S^{3/2}}).$$

Now it can be shown by (straightforward, but somewhat tedious) standard calculus that this function $\Delta^*(S)$ is strictly decreasing for $S \geq 145$, and that it tends to 4.5, as S tends to infinity. Hence, for $S \geq 8061$ and $S(S - 6\sqrt{S}) < Q \leq S^2$ the greatest possible difference between the upper bound and the lower bound in (5) equals $\Delta^*(8061) = 4.9999669 < 5$. This leaves space for at most five integer values between the two bounds on ξ , exactly as claimed in the lemma. Finally, we note that $S^2 - 12S^{3/2} > (S - 8\sqrt{S})^2$ holds for all $S \geq 8061$, and that

$$\frac{1}{2}(S + \sqrt{2Q - S^2}) > \frac{1}{2}(S + (S - 8\sqrt{S})) = S - 4\sqrt{S}.$$

Together with (5), this now yields $\xi \geq S - 4\sqrt{S}$ and completes the proof. $\square$

3 The Algorithm

We apply the results of Section 2 to get a polynomial time algorithm for Fourier's general problem. Hence, let S and Q be two positive integers that constitute an input to this problem.

1. If $S \leq 8060$, then solve the problem by complete enumeration. STOP.
2. If $Q < S$, or if $Q > S^2$, or if Q and S are of different parity, then output NO and STOP.

3. If $S \leq Q \leq S(S - 6\sqrt{S})$, then output YES and STOP.
4. If $S \geq 8061$ and $S(S - 6\sqrt{S}) < Q \leq S^2$, then determine all integers ξ that satisfy $\frac{1}{2}(S + \sqrt{2Q - S^2}) \leq \xi \leq \sqrt{Q}$. For each such ξ , solve the instance $(S - \xi, Q - \xi^2)$ recursively. Output YES if and only if at least one of these instances is a YES-instance.

Observation 2, Lemma 3, and Lemma 4 yield the correctness of Step 2, of Step 3, and of Step 4 of this algorithm, respectively.

Let $T(S)$ denote the maximum running time of the algorithm on all the instances (S, Q) with $Q \leq S^2$. The algorithm only performs elementary arithmetical operations on integers with $O(\log S)$ bits, like addition, multiplication, division, evaluation of square-roots, etc. It is safe to assume that each such operation can be performed in $O(\log^2 S)$ time; see for instance Alt [1] for a discussion of these issues. By Lemma 4, whenever the algorithm enters Step 4, it makes at most five recursive calls for new instances with $S^{\text{new}} = S - \xi \leq 4\sqrt{S}$. Thus, the time complexity $T(S)$ satisfies

$$T(S) \leq 5 \cdot T(4\sqrt{S}) + O(\log^2 S). \quad (6)$$

It is routine to deduce from (6) that $T(S) = O(\log^c S)$ for any $c > \log_2 5 \approx 2.33$. We summarize the main result of this note.

Theorem 5 *For positive integers S and Q , we can determine in polynomial time $O(\log^{2.33} S)$ whether there exist positive integers $x_1, \dots, x_k$ with $\sum_{i=1}^k x_i = S$ and $\sum_{i=1}^k x_i^2 = Q$. $\square$*

References

1. H. ALT (1978/79). Square rooting is as difficult as multiplication. *Computing* 21, 221–232.
2. J. HERIVEL (1975). *Joseph Fourier – The Man and the Physicist*. Clarendon Press, Oxford.
3. B. TURNER (1980). Fourier’s seventeen lines problem. *Mathematics Magazine* 53, 217–219.

Jacobi Curves: Computing the Exact Topology of Arrangements of Non-singular Algebraic Curves^{*}

Nicola Wolpert

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
nicola@mpi-sb.mpg.de

Abstract. We present an approach that extends the Bentley-Ottmann sweep-line algorithm [2] to the exact computation of the topology of arrangements induced by non-singular algebraic curves of arbitrary degrees. Algebraic curves of degree greater than 1 are difficult to handle in case one is interested in exact and efficient solutions. In general, the coordinates of intersection points of two curves are not rational but algebraic numbers and this fact has a great negative impact on the efficiency of algorithms coping with them. The most serious problem when computing arrangements of non-singular algebraic curves turns out to be the detection and location of tangential intersection points of two curves. The main contribution of this paper is a solution to this problem, using only rational arithmetic. We do this by extending the concept of Jacobi curves introduced in [11]. Our algorithm is output-sensitive in the sense that the algebraic effort we need for sweeping a tangential intersection point depends on its multiplicity.

1 Introduction

Computing arrangements of curves is one of the fundamental problems in computational geometry and algebraic geometry. For arrangements of lines defined by rational numbers all computations can be done over the field of rational numbers avoiding numerical errors and leading to exact mathematical results.

As soon as higher degree algebraic curves are considered, instead of linear ones, things become more difficult. In general, the intersection points of two planar curves defined by rational polynomials have irrational coordinates. That means instead of rational numbers one now has to deal with algebraic numbers. One way to overcome this difficulty is to develop algorithms that use floating point arithmetic. These algorithms are quite fast but in degenerate situations they can lead to completely wrong results because of approximation errors, rather

^{*} Partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG – Effective Computational Geometry for Curves and Surfaces)

than just slightly inaccurate outputs. Assume that for two planar curves one is interested in the number of intersection points. If the curves have tangential intersection points, the slightest inaccuracy can lead to a wrong output.

A second approach besides using floating point arithmetic is to use exact algebraic computation methods like the use of the gap theorem [4] or multivariate Sturm sequences [16]. Then of course the results are correct, but the algorithms in general are very slow.

We consider arrangements of non-singular curves in the real plane defined by rational polynomials. Although the non-singularity assumption is a strong restriction on the curves we consider, this class of curves is worthwhile to be studied because of the general nature of the main problem that has to be solved. Two algebraic curves can have tangential intersections and it is inevitable to determine them precisely in the case we are interested in exact computation. As a main tool for solving this problem we will introduce generalized Jacobi curves, for more details consider [22]. Our resulting algorithm computes the exact topology using only rational arithmetic. It is output-sensitive in the sense that the algebraic degree of the Jacobi curve that is constructed to locate a tangential intersection point depends on its multiplicity.

2 Previous Work

As mentioned, methods for the calculation of arrangements of algebraic curves are an important area of research in computational geometry. A great focus is on arrangements of linear objects. Algorithms coping with linear primitives can be implemented using rational arithmetic, leading to exact mathematical results in any case. For fast filtered implementations see for example the ones in LEDA [15] and CGAL [10]. There are also some geometric methods dealing with arbitrary curves, for example [1], [7], [18], [20]. But all of them neglect the problem of exact computation in the way that they are based on an idealized real arithmetic provided by the real RAM model of computation. The assumption is that all, even irrational, numbers are representable and that one can deal with them in constant time. This postulate is not in accordance with real computers.

Recently the exact computation of arrangements of non-linear objects has come into the focus of research. Wein [21] extended the CGAL implementation of planar maps to conic arcs. Berberich et al. [3] made a similar approach for conic arcs based on the improved LEDA [15] implementation of the Bentley-Ottmann sweep-line algorithm [2]. For conic arcs the problem of tangential intersection points is not serious because the coordinates of every such point are one-root expressions of rational numbers. Eigenwillig et al. [9] extended the sweep-line approach to cubic arcs. All tangential intersection points in the arrangements of cubic arcs either have coordinates that are one-root expressions or they are of multiplicity 2 and therefore can be solved using the Jacobi curve introduced in [11].

Arrangements of quadric surfaces in $\mathbb{R}^3$ are considered by Wolpert [22], Dupont et al. [8], and Mourrain et al. [17]. By projection the first author re-

duces the spatial problem to the one of computing planar arrangements of algebraic curves of degree at most 4. The second authors directly work in space determining a parameterization of the intersection curve of two arbitrary implicit quadrics. The third approach is a space sweep. Here the main task is to maintain the planar arrangements of conics on the sweep-plane.

For computing planar arrangements of arbitrary planar curves very little is known. An exact approach using rational arithmetic to compute the topological configuration of a single curve is done by Sakkalis [19]. Hong improves this idea by using floating point interval arithmetic [13]. For computing arrangements of curves we are also interested in intersection points of two or more curves. Of course we could interpret these points as singular points of the curve that is the union of both. But this would unnecessarily increase the degree of the algebraic curves we consider and lead to slow computation.

MAPC [14] is a library for exact computation and manipulation of algebraic points. It includes a package for determining arrangements of planar curves. For degenerate situations like tangential intersections the use of the gap theorem [4] or multivariate Sturm sequences [16] is proposed. Both methods are not efficient.

3 Notation

The objects we consider and manipulate in our work are non-singular algebraic curves represented by rational polynomials. We define an *algebraic curve* in the following way: Let f be a polynomial in $\mathbb{Q}[x, y]$. We set $\text{ZERO}(f) := \{(\alpha, \beta) \in \mathbb{R}^2 \mid f(\alpha, \beta) = 0\}$ and call $\text{ZERO}(f)$ the *algebraic curve* defined by f . If the context is unambiguous, we will often identify the defining polynomial of an algebraic curve with its zero set.

For an algebraic curve f we define its *gradient vector* to be $\nabla f := (f_x, f_y) \in (\mathbb{Q}[x, y])^2$ with $f_x := \frac{\partial f}{\partial x}$. We assume the set of input curves to be *non-singular*, that means for every point $(\alpha, \beta) \in \mathbb{R}^2$ with $f(\alpha, \beta) = 0$ we have $(\nabla f)(\alpha, \beta) = (f_x(\alpha, \beta), f_y(\alpha, \beta)) \neq (0, 0)$. A point (α, β) with $(\nabla f)(\alpha, \beta) = (0, 0)$ we would call *singular*. The geometric interpretation is that for every point (α, β) of f there exists a unique tangent line to the curve f . This tangent line is perpendicular to $(\nabla f)(\alpha, \beta)$. From now on we assume that all curves we consider are non-singular.

We call a point $(\alpha, \beta) \in \mathbb{R}^2$ of f *extreme* if $f_y(\alpha, \beta) = 0$. Extreme points have a vertical tangent. A point $(\alpha, \beta) \in \mathbb{R}^2$ of f is named a *flex* if the curvature of f becomes zero in (α, β) : $0 = (f_{xx}f_y^2 - 2f_xf_yf_{xy} + f_{yy}f_x^2)(\alpha, \beta)$.

Two curves f and g have a *disjoint factorization* if they only share a common constant factor. Without loss of generality we assume that this is the case for every pair of curves f and g we consider during our computation. Disjoint factorization can be easily tested and established by a bivariate gcd-computation.

For two curves f and g a point (α, β) in the real plane is called an *intersection point* if it lies on f as well as on g . It is called a *tangential intersection point* of f and g if additionally the two gradient vectors are linearly dependend: $(f_xg_y - f_yg_x)(\alpha, \beta) = 0$. Otherwise we speak of a *transversal intersection point*.

Last but not least we will name some properties of curves that are, unlike the previous definitions, not intrinsic to the geometry of the curves but depend on our chosen coordinate system.

We call a single curve $f = f_n(x) \cdot y^n + f_{n-1}(x) \cdot y^{n-1} + \dots + f_0(x) \in \mathbb{Q}[x, y]$ *generally aligned* if $f_n(x) = \text{constant} \neq 0$, in which case f has no vertical asymptotes. Two curves f and g are termed to be in *general relation* if every two common roots $(\alpha_1, \beta_1) \neq (\alpha_2, \beta_2) \in \mathbb{C}^2$ of f and of g have different x -values $\alpha_1 \neq \alpha_2$.

Next we will introduce the notation of *well-behavedness* of a pair of curves. We will first give the formal definition and then describe the geometric intuition behind. We say that two pairs of curves (f_1, g_1) and (f_2, g_2) are *separate* if

1. either there are non-zero constants c_1, c_2 with $f_1 = c_1 \cdot f_2$ and $g_1 = c_2 \cdot g_2$
2. or the x -values of the complex roots of f_1 and g_1 differ pairwise from the x -values of the complex roots of f_2 and g_2 .

We call two curves f and g *well-behaved* if

1. f and g are both generally aligned,
2. f and g are in general relation, and
3. the pairs of curves (f, g) , (f, f_y) , and (g, g_y) are pairwise separate.

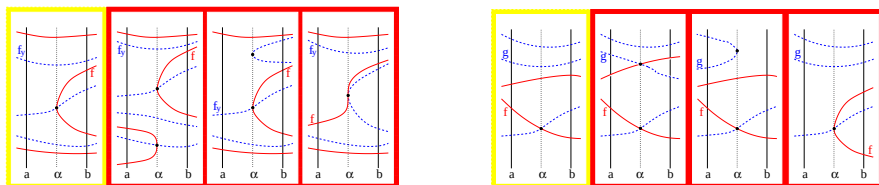


Fig. 1. In the leftmost box of the left picture the curves f and f_y are well-behaved, in the following three boxes they are not. In the leftmost box of the right picture the curves f and g are well-behaved, in the following three boxes they are not.

We will shortly give an idea of what well-behavedness of two curves means. Let (α, β) be an intersection point of two curves f and g . We first consider the case $g = f_y$ (left picture in Figure 1). If f and f_y are well-behaved, there exists a vertical stripe $a \leq x \leq b$ with $a < \alpha < b$ such that (α, β) is the only extreme point of f inside the stripe and the stripe contains no extreme point of f_y (and no singular point of f_y). Especially this means that flexes of f do not have a vertical tangent.

Next consider the case that neither f nor g is a constant multiple of the partial derivative of the other (right picture in Figure 1): there are no constants c_1, c_2 with $f = c_1 \cdot g_y$ or $g = c_2 \cdot f_y$. If f and g are well-behaved, then there exists a vertical stripe $a \leq x \leq b$ with $a < \alpha < b$ that contains exactly one intersection

point of f and g , namely (α, β) , and there is no extreme point of f or g inside this stripe. Especially this means that f and g do not intersect in extreme points.

A random shear at the beginning will establish well-behaved input-curves with high probability. We can test whether a pair of curves is well-behaved by gcd-, resultant-, and subresultant-computation. Due to the lack of space we omit the details. If we detect during the computation that the criterion is not fulfilled for one pair of curves, then we know that we are in a degenerate situation due to the choice of our coordinate system. In this case we stop, shear the whole set of input curves by random (for a random $v \in \mathbb{Q}$ we apply the affine transformation $\psi(x, y) = (x + vy, y)$ to each input polynomial) and restart from the beginning. A shear does not change the topology of the arrangement and we end up with pairs of well-behaved curves.

4 The Overall Approach

We are interested in the topology of a planar arrangement of a set F of n non-singular input curves. The curves partition the affine space in a natural way into three different types of maximal connected regions of dimensions 2, 1, and 0 called faces, edges, and vertices, respectively.

We want to compute the arrangement with a sweep-line algorithm. At each time during the sweep the branches of the curves intersect the sweep-line in some order. While moving the sweep-line along the x -axis a change in the topology of the arrangement takes place if this ordering changes. This happens at intersection points of at least two different curves $f, g \in F$ and at extreme points of a curve $f \in F$. At extreme points geometrically two new branches of f start or two branches of f end. Extreme points of f are intersection points of f and f_y . This leads to the following definition of points on the x -axis that force the sweep-line to stop and to recompute the ordering of the curves:

Definition 1 *The event points of a planar arrangement induced by a set F of non-singular planar curves are defined as the intersection points of each two curves $f, g \in F$ and as the intersection points of f and f_y for all $f \in F$.*

Our main algorithmic approach follows the ideas of the Bentley-Ottmann sweep [2]. We hold up an X - and a Y -structure. The X -structure contains the x -coordinates of event points. In the Y -structure we maintain the ordering of the curves along the sweep-line. At the beginning we found that for every $f \in F$ the curves f and f_y are well-behaved. We insert the x -coordinates of all extreme points into the empty X -structure. We shortly remark that there can be event points left to the leftmost extreme point. This can be resolved by moving the sweep-line to the left until all pairs of adjacent curves in the Y -structure have their intersection points to the right.

If the sweep-line reaches the next event point we stop, identify the pairs of curves that intersect, the kind of intersection they have and their involved branches, recompute the ordering of the curves along the sweep-line, and according to this we update the Y -structure. If two curves become adjacent that

were not adjacent in the past, we test whether they are well-behaved. If f and g are not well-behaved, we shear the whole arrangement and start from the beginning. Otherwise we compute the x -coordinates of their intersection points and insert them into the X -structure.

5 The X -Structure

In order to make the overall approach compute the exact mathematical result in every case there are some problems that have to be solved. Describing the sweep we stated that one of the fundamental operations is the following: For two well-behaved curves f and g insert the x -coordinates of their intersection points into the X -structure. A well known algebraic method is the resultant computation of f and g with respect to y [6]. We can compute a polynomial $\text{res}(f, g) \in \mathbb{Q}[x]$ of degree at most $\deg(f) \cdot \deg(g)$ with the following property:

Proposition 1 *Let $f, g \in \mathbb{Q}[x, y]$ be generally aligned curves that are in general relation. A number $\alpha \in \mathbb{R}$ is a root of $\text{res}(f, g)$ if and only if there exists exactly one $\beta \in \mathbb{C}$ such that $f(\alpha, \beta) = g(\alpha, \beta) = 0$ and $\beta \in \mathbb{R}$.*

The x -coordinates of real intersection points of f and g are exactly the real roots of the resultant polynomial $\text{res}(f, g)$. Unfortunately, the intersection points of algebraic curves in general have irrational coordinates. By definition, every root of $\text{res}(f, g)$ is an *algebraic number*. For $\deg(\text{res}(f, g)) > 2$ there is no general way via radicals to explicitly compute the algebraic numbers in every case. But we can determine an *isolating interval* for each real root α of $\text{res}(f, g)$, for example with the algorithm of Uspensky [5]. We compute two rational numbers a and b such that α is the one and only real root of $\text{res}(f, g)$ in $[a, b]$. The pair $(\text{res}(f, g), [a, b])$ yields a non-ambiguous rational representation of α . Of course in this representation the entry $\text{res}(f, g)$ could be exchanged by any rational factor $p \in \mathbb{Q}[x]$ of $\text{res}(f, g)$ with $p(\alpha) = 0$. Additionally we like α to remember the two curves f and g it originates from. We end up with inserting a representation $(p, [a, b], f, g)$ for every event point induced by f and g into the X -structure. Remark that several pairs of curves can intersect at the event point $x = \alpha$. In this case there are several representations of the algebraic number α in the X -structure, one for each pair of intersecting curves.

During the sweep we frequently have to determine the next coming event point. In order to support this query with the help of the isolating intervals we finally have to ensure the following invariant: Every two entries in the X -structure either represent the same algebraic number, and in this case the isolating intervals in their representation are identical, or their isolating intervals are disjoint. The invariant can be easily established and maintained using gcd-computation of the defining univariate polynomials and bisection by midpoints of the isolating intervals.

6 The Y-Structure

A second problem that has to be solved is how to update the Y -structure at an event point. At an event point we have to stop with the sweep-line, identify the pairs of curves that intersect and their involved branches, and recompute the ordering of the curves along the sweep-line. As we have seen, the x -coordinate α of an event point is represented by at least one entry of the form $(p, [a, b], f, g)$ in the x -structure. So we can directly determine the pairs of curves that intersect at $x = \alpha$. For each pair f and g of intersecting curves we have to determine their involved branches. Furthermore we have to decide whether these two branches cross or just touch, but do not cross each other. As soon as we have these two information, updating the ordering of the curves along the sweep-line is easy.

In general, event points have irrational coordinates and therefore we cannot exactly stop the sweep-line at $x = \alpha$. The only thing we can do is stopping at the rational point a to the left of α and at the rational point b to the right of α . Using a root isolation algorithm, gcd-computation of univariate polynomials, and bisection by midpoints of the separating intervals we compute the sequence of the branches of f and g along the rational line $x = a$. We do the same along the line $x = b$. Finally, we compare these two orderings. In some cases this information is sufficient to determine the kind of event point and the involved branches of the curves inside the stripe $a \leq x \leq b$. Due to our assumption of well-behavedness we can directly compute extreme points of f (consider the left picture in Figure 2):

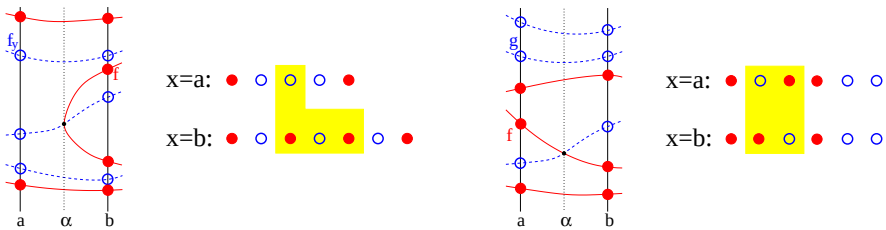


Fig. 2. For computing extreme points it is sufficient to compare the sequence of f and f_y at $x = a$ to the left and at $x = b$ to the right of α (left picture). The same holds for computing intersection points of odd multiplicity of two curves f and g (right picture).

Theorem 1 *Let $(\alpha, \beta) \in \mathbb{R}^2$ be an extreme point of a non-singular curve f and assume that f and f_y are well-behaved. We can compute two rational numbers $a \leq \alpha \leq b$ with the following property: the identification of the involved branches of f is possible by just comparing the sequence of hits of f and f_y along $x = a$ and along $x = b$.*

Proof. (Sketch) By assumption the curves f and f_y are well-behaved and therefore we know that α is not an extreme or singular point of f_y . We shrink

the isolating interval $[a, b]$ of α until it contains no real root of $\text{res}(f_y, f_{yy}, y)$. Afterwards the number and ordering of the branches of f_y does not change in the interval $[a, b]$. The number of branches of f at $x = a$ differs by 2 from the one at $x = b$. At $x = a$ at least one branch of f_y lies between two branches of f . The same holds at $x = b$.

Using root isolation we compare from $-\infty$ upwards the sequences of roots of f and f_y at $x = a$ and at $x = b$. The branch i of f that causes the first difference (either at $x = a$ or at $x = b$) intersects the $(i + 1)$ -st branch of f in an extreme point. $\square$

The same idea can be used to compute intersection points of odd multiplicity between two curves f and g where two branches of f and g cross each other (see the right picture in Figure 2) because we have an observable transposition in the sequences. Of course the test can be easily extended to arbitrary curves under the assumption that the intersection point (α, β) is not a singular point of any of the curves.

What remains to do is locating intersection points (α, β) of even multiplicity. These points are rather difficult to locate. From the information how the curves behave slightly to the left and to the right of the intersection point we cannot draw any conclusions. At $x = a$ and at $x = b$ the branches of f and g appear in the same order, see Figure 3. We will show in the next section how to extend the idea of Jacobi curves introduced in [11] to intersection points of arbitrary multiplicity.

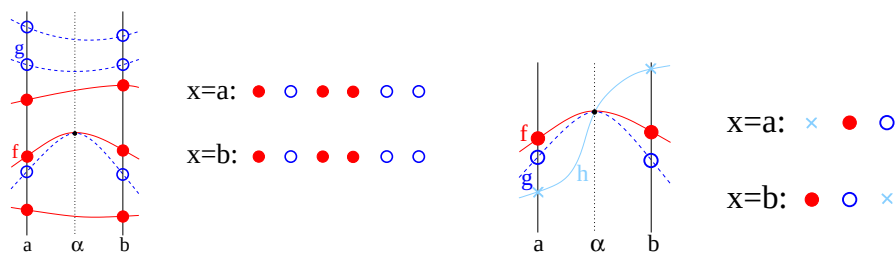


Fig. 3. Intersection points of even multiplicity lead to the same sequence of f and g to the left and to the right of α . Introduce an auxiliary curve h in order to locate these intersection points.

7 The Jacobi Curves

In order to locate an intersection point of even multiplicity between two curves f and g it would be helpful to know a third curve h that cuts f as well as g transversally in this point, see right picture in Figure 3. This would reduce the problem of locating the intersection point of f and g to the easy one of locating the transversal intersection point of f and h and the transversal intersection

point of g and h . In the last section we have shown how to compute the indices i , j , and k of the intersecting branches of f , g , and h , respectively. Once we have determined these indices we can conclude that the i th branch of f intersects the j th branch of g .

We will give a positive answer to the existence of transversal curves with the help of the Theorem of Implicit Functions. Let $(\alpha, \beta) \in \mathbb{R}^2$ be a real intersection point of $f, g \in \mathbb{Q}[x, y]$. We will iteratively define a sequence of polynomials $\tilde{h}_1, \tilde{h}_2, \tilde{h}_3, \dots$ such that $\tilde{h}_k$ cuts transversally through f in (α, β) for some index k . If f and g are well-behaved, the index k is equal to the degree of α as a root of $\text{res}(f, g, y)$. The result that introducing an additional curve can solve tangential intersections is already known for $k = 2$ [11]. What is new is that this concept can be extended to every multiplicity $k > 2$. All the following results are not restricted to non-singular curves. We will show in Theorem 2 that we can determine every tangential intersection point of two arbitrary curves provided that it is not a singular point of one of the curves.

Definition 2 *Let f and g be two planar curves. We define generalized Jacobi curves in the following way:*

$$\tilde{h}_1 := g, \quad \tilde{h}_{i+1} := (\tilde{h}_i)_x f_y - (\tilde{h}_i)_y f_x.$$

Theorem 2 *Let f and g be two algebraic curves with disjoint factorizations. Let (α, β) be an intersection point of f and g that neither is a singular point of f nor of g . There exists an index $k \geq 1$ such that $\tilde{h}_k$ cuts transversally through f in (α, β) .*

Proof. In the case g cuts through f in the point (α, β) , especially if (α, β) is a transversal intersection point of f and g , this is of course true for $\tilde{h}_1 = g$. So assume in the following that $(g_x f_y - g_y f_x)(\alpha, \beta) = \tilde{h}_2(\alpha, \beta) = 0$. From now on we will only consider the polynomials $\tilde{h}_i$ with $i \geq 2$.

By assumption every point (α, β) is a non-singular point of f : $(f_x, f_y)(\alpha, \beta) \neq 0$. We only consider the case $f_y(\alpha, \beta) \neq 0$. In the case $f_x(\alpha, \beta) \neq 0$ and $f_y(\alpha, \beta) = 0$ we would proceed the same way as described in the following by just exchanging the two variables x and y . The property $f_y(\alpha, \beta) \neq 0$ leads to $(\frac{f_x}{f_y} g_y)(\alpha, \beta) = g_x(\alpha, \beta)$ and because $(g_x, g_y)(\alpha, \beta) \neq (0, 0)$ we conclude $g_y(\alpha, \beta) \neq 0$. From the Theorem of Implicit Functions we derive that there are real open intervals $I_x, I_y \subset \mathbb{R}$ with $(\alpha, \beta) \in I_x \times I_y$ such that

1. $f_y(x, y_0) \neq 0$ and $g_y(x_0, y_0) \neq 0$ for all $(x_0, y_0) \in I_x \times I_y$,
2. there exist continuous functions $F, G : I_x \rightarrow I_y$ with the two properties
 - a) $f(x, F(x)) = g(x, G(x)) = 0$ for all $x \in I_x$
 - b) $(x, y) \in I_x \times I_y$ with $f(x, y) = 0$ leads to $y = F(x)$,
 $(x, y) \in I_x \times I_y$ with $g(x, y) = 0$ leads to $y = G(x)$.

Locally around the point (α, β) the curve defined by the polynomial f is equal to the graph of the function F . The same holds for g and G . Especially

we have $\beta = F(\alpha) = G(\alpha)$. Moreover, the Theorem of Implicit Holomorphic Functions implies that F as well as G are holomorphic and thus developable in a Taylor series around the point (α, β) [12].

In the following we will sometimes consider the functions $h_i : I_x \times I_y \rightarrow \mathbb{R}$, $i \geq 2$, with

$$h_2 := \frac{g_x}{g_y} - \frac{f_x}{f_y} = \frac{\tilde{h}_2}{g_y f_y}, \quad h_{i+1} := (h_i)_x - (h_i)_y \cdot \frac{f_x}{f_y}$$

instead of the polynomials $\tilde{h}_i$. Each h_i is well defined for $(x, y) \in I_x \times I_y$. We have the following relationship between the functions h_i and the polynomials $\tilde{h}_i$ defined before: For each $i \geq 2$ there exist functions $\delta_{i,2}, \delta_{i,3}, \dots, \delta_{i,i} : I_x \times I_y \rightarrow \mathbb{R}$ such that

$$(*) \quad h_i = \delta_{i,2} \cdot \tilde{h}_2 + \delta_{i,3} \cdot \tilde{h}_3 + \dots + \delta_{i,i} \cdot \tilde{h}_i$$

with $\delta_{i,i}(x, y) \neq 0$ for all $(x, y) \in I_x \times I_y$. For $i = 2$ this is obviously true with $\delta_{2,2} = (g_y f_y)^{-1}$. The general case follows by induction on i .

Let us assume we know the following proposition: Let $k \geq 1$. If $F^{(i)}(\alpha) = G^{(i)}(\alpha)$ for all $0 \leq i \leq k-1$, then $h_{k+1}(\alpha, \beta) = G^{(k)}(\alpha) - F^{(k)}(\alpha)$.

We know that the two polynomials f and g have disjoint factorizations. That means the Taylor series of F and G differ in some term. Remember that we consider the case that the curves defined by f and g intersect tangentially in the point (α, β) . So there is an index $k \geq 2$ such that $F^{(i)}(\alpha) = G^{(i)}(\alpha)$ for all $0 \leq i \leq k-1$ and $F^{(k)}(\alpha) \neq G^{(k)}(\alpha)$. According to the proposition we have $h_{i+1}(\alpha, \beta) = G^{(i)}(\alpha) - F^{(i)}(\alpha) = 0$ for all $1 \leq i \leq k-1$. From equation $(*)$ we inductively obtain also $\tilde{h}_{i+1}(\alpha, \beta) = 0$, $1 \leq i \leq k-1$. Especially this means that $\tilde{h}_k$ intersects f and g in (α, β) . The intersection is transversal if and only if $((\tilde{h}_k)_x f_y - (\tilde{h}_k)_y f_x)(\alpha, \beta) = \tilde{h}_{k+1}(\alpha, \beta) \neq 0$. This follows easily from $0 \neq G^{(k)}(\alpha) - F^{(k)}(\alpha) = h_{k+1}(\alpha, \beta) = \delta_{k+1,k+1}(\alpha, \beta) \cdot \tilde{h}_{k+1}(\alpha, \beta)$. $\square$

It remains to state and prove the proposition:

Proposition 2 *Let $k \geq 1$. If $F^{(i)}(\alpha) = G^{(i)}(\alpha)$ for all $0 \leq i \leq k-1$, then $h_{k+1}(\alpha, \beta) = G^{(k)}(\alpha) - F^{(k)}(\alpha)$.*

Proof. For each $i \geq 2$ we define a function $H_i : I_x \rightarrow \mathbb{R}$ by $H_i(x) := h_i(x, F(x))$. For $x = \alpha$ we derive $H_i(\alpha) = h_i(\alpha, \beta)$. So in terms of our new function we want to prove that $H_{k+1}(\alpha) = G^{(k)}(\alpha) - F^{(k)}(\alpha)$ holds if $F^{(i)}(\alpha) = G^{(i)}(\alpha)$ for all $0 \leq i \leq k-1$.

By definition we have $f(x, F(x)) : I_x \rightarrow \mathbb{R}$ and $f(x, F(x)) = 0$ for all $x \in I_x$. That means $f(x, F(x))$ is constant and therefore $0 = f(x, F(x))' = f_x(x, F(x)) + F'(x) f_y(x, F(x))$. We conclude $F'(x) = -f_x(x, F(x)) / f_y(x, F(x))$ and this directly leads to the equality $H'_i(x) = H_{i+1}(x)$. Inductively we obtain $H_{i+1}(x) = H_2^{(i-1)}(x)$ for all $i \geq 1$. In order to prove the proposition it is sufficient to show the following: Let $k \geq 1$. If for all $0 \leq i \leq k-1$ we have $F^{(i)}(\alpha) = G^{(i)}(\alpha)$, then $H_2^{(k-1)}(\alpha) = (G' - F')^{(k-1)}(\alpha)$.

1. Let $k = 1$. Our assumption is $F(\alpha) = G(\alpha)$ and we have to show $H_2(\alpha) = (G' - F')(\alpha)$. We have

$$(**) \quad H_2(x) = h_2(x, F(x)) = \frac{g_x(x, F(x))}{g_y(x, F(x))} - \frac{f_x(x, F(x))}{f_y(x, F(x))}$$

$$\text{and } (G' - F')(x) = \frac{g_x(x, G(x))}{g_y(x, G(x))} - \frac{f_x(x, F(x))}{f_y(x, F(x))}$$

and both functions just differ in the functions that are substituted for y in $\frac{g_x(x, y)}{g_y(x, y)}$. In the equality of $H_2(x)$ we substitute $F(x)$, whereas in the one of $(G' - F')$ we substitute $G(x)$. But of course $F(\alpha) = G(\alpha)$ leads to $H_2(\alpha) = (G' - F')(\alpha)$.

2. Let $k > 1$. We know that $F^{(i)}(\alpha) = G^{(i)}(\alpha)$ for all $0 \leq i \leq k - 1$. We again use the equations $(**)$ and the fact that $H_2(x)$ and $(G' - F')$ only differ in the functions that are substituted for y in $\frac{g_x(x, y)}{g_y(x, y)}$.

By taking $(k - 1)$ times the derivative of $H_2(x)$ and $(G' - F')$, we structurally obtain the same result for both functions. The only difference is that some of the terms $F^{(i)}(x)$, $0 \leq i \leq k - 1$, in H_2 are exchanged by $G^{(i)}(x)$ in $(G' - F')$. But due to our assumption we have $F^{(i)}(\alpha) = G^{(i)}(\alpha)$ for all $0 \leq i \leq k - 1$ and we obtain $H_2^{(k-1)}(\alpha) = (G' - F')^{(k-1)}(\alpha)$. $\square$

We have proven that for a non-singular tangential intersection point of f and g there exists a curve $\tilde{h}_k$ that cuts both curves transversally in this point. The index k depends on the degree of similarity of the functions that describe both polynomials in a small area around the given point. The degree of similarity is measured by the number of successive matching derivatives in this point. An immediate consequence of the previous theorem, together with the well known fact that the resultant of two univariate polynomials equals the product of the differences of their roots [6], is that we can obtain the index k by just looking at the resultant of f and g :

Corollary 1 *Let $f, g \in \mathbb{Q}[x, y]$ be two polynomials in general relation and let (α, β) be a non-singular intersection point of the curves defined by f and g . If k is the degree of α as a root of the resultant $\text{res}(f, g, y)$, then $\tilde{h}_k$ cuts transversally through f . (proof omitted)*

Acknowledgements. The author would like to thank Elmar Schömer and Raimund Seidel for useful discussions and suggestions and Arno Eigenwillig for carefully proof-reading the paper.

References

1. C. Bajaj and M. S. Kim. Convex hull of objects bounded by algebraic curves. *Algorithmica*, 6:533–553, 1991.
2. J. L. Bentley and T. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

3. E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, and E. Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In *ESA 2002, Lecture Notes in Computer Science*, pages 174–186, 2002.
4. J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1987.
5. G. E. Collins and R. Loos. Real zeros of polynomials. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra: Symbolic and Algebraic Computation*, pages 83–94. Springer-Verlag, New York, NY, 1982.
6. D. Cox, J. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms*. Springer, New York, 1997.
7. D. P. Dobkin and D. L. Souvaine. Computational geometry in a curved world. *Algorithmica*, 5:421–457, 1990.
8. L. Dupont, D. Lazard, S. Lazard, and S. Petitjean. Near-optimal parameterization of the intersection of quadrics. In *Proc. 19th Annu. ACM Sympos. Comput. Geom.*, pages 246–255, 2003.
9. A. Eigenwillig, E. Schömer, and N. Wolpert. Sweeping arrangements of cubic segments exactly and efficiently. Technical Report ECG-TR-182202-01, 2002.
10. E. Flato, D. Halperin, I. Hanniel, and O. Nechushtan. The design and implementation of planar maps in cgal. In *Proceedings of the 3rd Workshop on Algorithm Engineering*, Lecture Notes Comput. Sci., pages 154–168, 1999.
11. N. Geismann, M. Hemmer, and E. Schömer. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 264–271, 2001.
12. R. Gunning and H. Rossi. *Analytic functions of several complex variables*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1965.
13. H. Hong. An efficient method for analyzing the topology of plane real algebraic curves. *Mathematics and Computers in Simulation*, 42:571–582, 1996.
14. J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: A library for efficient and exact manipulation of algebraic points and curves. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 360–369, 1999.
15. K. Mehlhorn and S. Näher. *LEDA – A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
16. P. S. Milne. On the solutions of a set of polynomial equations. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 89–102. 1992.
17. B. Mourrain, J.-P. Tékourt, and M. Teillaud. Sweeping an arrangement of quadrics in 3d. In *Proceedings of 19th European Workshop on Computational Geometry*, 2003.
18. K. Mulmuley. A fast planar partition algorithm, II. *J. ACM*, 38:74–103, 1991.
19. T. Sakkalis. The topological configuration of a real algebraic curve. *Bulletin of the Australian Mathematical Society*, 43:37–50, 1991.
20. J. Snoeyink and J. Hershberger. Sweeping arrangements of curves. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 6:309–349, 1991.
21. R. Wein. On the planar intersection of natural quadrics. In *ESA 2002, Lecture Notes in Computer Science*, pages 884–895, 2002.
22. N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. Universität des Saarlandes, Saarbrücken, 2002. Ph.D. Thesis.

Streaming Geometric Optimization Using Graphics Hardware*

Pankaj K. Agarwal¹, Shankar Krishnan², Nabil H. Mustafa¹, and
Suresh Venkatasubramanian²

¹ Dept. of Computer Science, Duke University, Durham, NC 27708-0129, U.S.A.
{[pankaj](mailto:pankaj@cs.duke.edu), [nabil](mailto:nabil@cs.duke.edu)}@cs.duke.edu

² AT&T Labs – Research, 180 Park Ave, Florham Park, NJ 07932.
{[suresh](mailto:suresh@research.att.com), [krishnas](mailto:krishnas@research.att.com)}@research.att.com

Abstract. In this paper we propose algorithms for solving a variety of geometric optimization problems on a stream of points in $\mathbb{R}^2$ or $\mathbb{R}^3$. These problems include various extent measures (e.g. diameter, width, smallest enclosing disk), collision detection (penetration depth and distance between polytopes), and shape fitting (minimum width annulus, circle/line fitting). The main contribution of this paper is a *unified* approach to solving all of the above problems efficiently using modern graphics hardware. All the above problems can be approximated using a constant number of passes over the data stream. Our algorithms are easily implemented, and our empirical study demonstrates that the running times of our programs are comparable to the best implementations for the above problems. Another significant property of our results is that although the best known implementations for the above problems are quite different from each other, our algorithms all draw upon the same set of tools, making their implementation significantly easier.

1 Introduction

The study of *streaming data* is motivated by numerous applications that arise in the context of dealing with massive data sets. In this paper we propose algorithms for solving a variety of geometric optimization problems over a stream of two or three dimensional geometric data (e.g. points, lines, polygons). In particular, we study three classes of problems: (a) *Extent measures*: computing various *extent measures* (e.g. diameter, width, smallest enclosing circle) of a stream of points in $\mathbb{R}^2$ or $\mathbb{R}^3$, (b) *Collision detection*: computing the penetration depth of a pair of convex polyhedra in three dimensions and (c) *Shape fitting*: approximating a set of points by simple shapes like circles or annuli.

Many of the problems we study can be formulated as computing and/or overlaying lower and upper envelopes of certain functions. We will be considering approximate solutions, and thus it suffices to compute the value of these

* Pankaj Agarwal and Nabil Mustafa are supported by NSF grants ITR-333-1050, EIA-9870724, EIA-997287, and CCR-02-04118, and by a grant from the U.S.-Israeli Binational Science Foundation.

envelopes at a set of uniformly sampled points, i.e., on a grid. This allows us to exploit recent developments in graphics hardware accelerators. Almost all modern graphics cards (examples include the nVidia GeForce and ATI Radeon series) provide hardware support for computing the envelope of a stream of bivariate functions at a uniform sample of points in $[-1, +1]^2$ and for performing various arithmetic and logical operations on each of these computed values, which makes them ideal for our applications. We therefore study the above streaming problems in the context of graphics hardware.

Related work. In the standard streaming model the input $\{x_1, \dots, x_n\}$ is written in sequence on an *input tape*. The algorithm has a *read head*, and in each pass, the read head makes one sequential scan over the input tape. The efficiency of an algorithm is measured in terms of the size of the working space, the number of passes, and the time it spends on performing the computation. There are numerous algorithms for computing properties of data streams, as well as various lower bounds on the resources required [20]. Data stream computations of geometric properties like the diameter, convex hull, and minimum spanning tree have also received recent attention [15, 11, 13, 6].

Traditionally, graphics hardware has been used for rendering three dimensional scenes. But the growing sophistication of graphics cards and their relatively low cost has led researchers to use their power for a variety of problems in other areas, and specially in the context of geometric computing [12, 19, 16]. Fournier and Fussell [7] were the first to study general stream computations on graphics cards; a recent paper [8] shows lower bounds on the number of passes required by hardware-based k^{th} -element selection operations, as well as showing the necessity of certain hardware functions in reducing the number of passes in selection from $\Omega(n)$ to $O(\log n)$.

There has been extensive work in computational geometry and computing extent measures and shape fitting [3]. The most relevant work in a recent result by Agarwal *et al.* [2] which presents an algorithm for computing a small size “core set” C of a given set S of points in $\mathbb{R}^d$ whose extent approximates the extent of S , yielding linear time approximations for computing the diameter and smallest spherical shell of a point set. Their algorithm can be adapted to the streaming model, in the sense that C can be computed by performing one pass over S , after which one can compute an ε -approximation of the desired extent measure in $1/\varepsilon^{O(1)}$ time using $1/\varepsilon^{O(1)}$ memory.

Our work. In this paper, we demonstrate a large class of geometric optimization problems that can be approximated efficiently using graphics hardware. A unifying theme of the problems that we solve is that they can be expressed in terms of minimizations over envelopes of bivariate functions.

Extent problems: We present hardware-based algorithms for computing the diameter and width (in two and three dimensions) and the smallest enclosing ball (in two dimensions) of a set of points. All the algorithms are approximate, and compute the desired answer in a constant number of passes. We note here that although the number of passes is more than one, each pass does not use any information from prior passes and the computation effectively runs in a single

pass. For reasons that will be made clear in Section 4, the graphics pipeline requires us to perform a series of passes that explore different regions of the search space.

In addition, the smallest bounding box of a planar point set can also be approximated in a constant number of passes; computing the smallest bounding box in three dimensions can be done in $1/\sqrt{\alpha-1}$ passes, where α is an approximation parameter.

Collision detection: We present a hardware-based algorithm for approximating the *penetration depth* between two convex polytopes. In general, our method can compute any inner product-based distance between any two convex polyhedra (intersecting or not). Our approach can also be used to compute the Minkowski sum of two convex polygons in two dimensions.

Shape fitting and other problems: We also present heuristics for a variety of shape-fitting problems in the plane: computing the minimum width annulus, best-fit circle, and best-fit line for a set of points, and computing the Hausdorff distance between two sets of points. Our methods are also applicable to many problems in *layered manufacturing* [17].

Experimental results: An important practical consequence of our unified approach to solving these problems is that all our implementations make use of the same underlying procedures, and thus a single implementation provides much of the code for all of the problems we consider. We present an empirical study that compares our algorithms to existing implementations for representative problems from the above list; in all cases we are comparable, and in many cases we are far superior to existing software-based implementations.

2 Preliminaries

The graphics pipeline. The graphics pipeline is primarily used as a rendering (or “drawing”) engine to facilitate interactive display of complex three-dimensional geometry. The input to the pipeline is a set of geometric primitives and images, which are *transformed* and *rasterized* at various stages of the pipeline to produce an array of *fragments*, that is “drawn” on a two-dimensional grid of pixels known as the *frame buffer*. The frame buffer is a collection of several individual dedicated buffers (color, stencil, depth buffers etc.). The user interacts with the pipeline via a standardized software interface (such as OpenGL or DirectX) that is designed to mimic the graphics subsystem. For more details, the reader may refer to the OpenGL programming guide [21].

Computing Envelopes. Let $F = \{f_1, \dots, f_n\}$ be a set of d -variate functions. The *lower envelope* of F is defined as $E_F^-(x) = \min_i f_i(x)$, and the *upper envelope* of F is defined as $E_F^+(x) = \max_i f_i(x)$. The projection of E_F^- (resp. E_F^+) is called the *minimization* (resp. *maximization*) diagram of S . Set $f_F^-(x)$ (resp. $f_F^+(x)$) to be the index of a function of F that appears on its lower (resp. upper) envelope. Finally, define $I_F(x) = E_F^+(x) - E_F^-(x)$. We will omit the subscript F when it is obvious from the context. If F is a family of piecewise-linear bivariate functions, we can compute E^-, E^+, f^-, f^+ for each pixel $x \in [-1, +1]^2$, using the graphics

hardware. We will assume that function $f_i(x)$ can be described accurately as a collection of triangles.

Computing E^- (E^+): Each vertex v_{ij} is assigned a color equal to its z-coordinate (depth) (or function value). The graphics hardware generates color values across the face of a triangle by performing bilinear interpolation of the colors at the vertices. Therefore, the color value at each pixel correctly encodes the function value. We disable the stencil test, set the depth test to min (resp. max). After rendering all the functions, the color values in the framebuffer contains their lower (resp. upper) envelope. In the light of recent developments in programming the graphics pipeline, nonlinear functions can be encoded as part of a shading language (or fragment program) to compute their envelopes as well.

Computing f^- (f^+): Each vertex v_{ij} of function f_i is assigned the color c_i (in most cases, c_i is determined by the problem). By setting the graphics state similar to the previous case, we can compute f^- and f^+ .

In many of the problems we address, we will compute envelopes of distance functions. That is, given a distance function $\delta(\cdot, \cdot)$ and a set $S = \{p_1, \dots, p_n\}$ of points in $\mathbb{R}^2$, we define $F = \{f_i(x) \equiv \delta(x, p_i) \mid 1 \leq i \leq n\}$, and we wish to compute the lower and upper envelopes of F . For the Euclidean metric, the graph of each f_i is a cone whose axis is parallel to the z-axis and whose sides are at an angle of $\pi/4$ to the xy -plane. For the square Euclidean metric, it is a paraboloid symmetric around a vertical line. Such surfaces can be approximated to any desired degree of approximation by triangulations ([12]).

Approximations. For purposes of computation, the two-dimensional plane is divided into pixels. This discretization of the plane makes our algorithms approximate by necessity. Thus, for a given problem, the cost of a solution is a function both of the algorithm and the screen resolution. We define a (α, g) -approximation algorithm to be one that provides a solution of cost at most α times the optimal solution, with a grid cell size of $g = g(I, \alpha)$, where I is the instance of the problem. This definition implies that different instances of the same problem may require different grid resolutions.

3 Gauss Maps and Duality

Let $S = \{p_1, \dots, p_n\}$ be a set of n points in $\mathbb{R}^d$. A direction in $\mathbb{R}^d$ can be represented by a unit vector $u \in \mathbb{S}^{d-1}$. For $u \in \mathbb{S}^{d-1}$, let $\hat{u}$ be its *central* projection, i.e., the intersection point of the ray $\overrightarrow{o\hat{u}}$ with the hyperplane $x_d = 1$ (resp. $x_d = -1$) if u lies in the positive (resp. negative) hemisphere.

For a direction u , we define the *extremal point* in direction u to be $\lambda(u, S) = \arg \max_{p \in S} \langle \hat{u}, p \rangle$, where $\langle \cdot, \cdot \rangle$ is the inner product. The *directional width* of S is $\omega(u, S) = \max_{p \in S} \langle \hat{u}, p \rangle - \min_{p \in S} \langle \hat{u}, p \rangle$. The *Gaussian map* of the convex hull of S is the decomposition of $\mathbb{S}^{d-1}$ into maximal connected regions so that the extremal point is the same for all directions within one region.

For a point $p = (p_1, \dots, p_d)$, we define its *dual* to be the hyperplane $p^* : x_d = p_1x_1 + \dots + p_{d-1}x_{d-1} + p_d$. Let $H = \{p^* \mid p \in S\}$ be the set of hyperplanes dual to the points in S . The following is easy to prove.

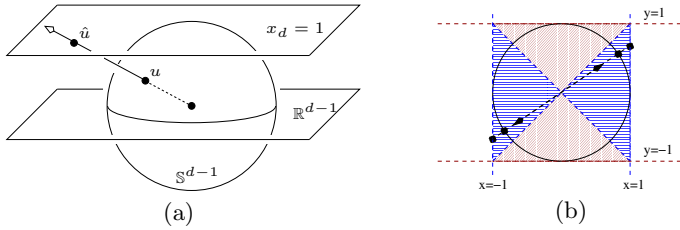


Fig. 1. (a) An illustration of central projection. (b) Two duals used to capture the Gaussian Map.

Lemma 1. For $u \in \mathbb{S}^{d-1}$, $\lambda(u, S) = f_H^+(\hat{u}_1, \dots, \hat{u}_{d-1})$ if u lies in the positive hemisphere, and $\lambda(u, S) = f_H^-(\hat{u}_1, \dots, \hat{u}_{d-1})$ if u lies in the negative hemisphere; here $\hat{u} = (\hat{u}_1, \dots, \hat{u}_d)$. $\square$

Hence, we can compute $\lambda(u, S)$ using f_H^+ and f_H^- . Note that the central projection of the portion of the Gaussian map of S in the upper (resp. lower) hemisphere is the maximization (resp. minimization) diagram of H . Thus, for $d = 3$ we can compute portion of the Gaussian map of S whose central projection lies in the square $[-1, +1]^2$, using graphics hardware, as described in Section 2. In other words, we can compute the extremal points of S for all u such that $\hat{u} \in [-1, 1]^2 \times \{1, -1\}$. If we also take the central projection of a vector $u \in \mathbb{S}^2$ onto the planes $y = 1$ and $x = 1$, then at least one of the central projections of u lies in the square $[-1, +1]^2$ of the corresponding plane. Let R_x (resp. R_y) be the rotation transform that maps the unit vector $(1, 0, 0)$ (resp. $(0, 1, 0)$) to $(0, 0, 1)$. Let H_x (resp. H_y) be the set of planes dual to the point set $R_x(S)$ (resp. $R_y(S)$). If we compute $f_{H_x}^+, f_{H_x}^-, f_{H_y}^+$, and $f_{H_y}^-$ for all $x \in [-1, +1]^2$, then we can guarantee that we have computed extremal points in all directions (see Fig. 1(b) for an example in two dimensions).

In general, vertices of the arrangement of dual hyperplanes may not lie in the box $[-1, +1]^3$. A generalization of the above idea can be used to compute a family of three duals such that any vertex of the dual arrangement is guaranteed to lie in the region $[-1, +1]^2 \times [-n, n]$ in *some* dual. Such a family of duals can be used to compute more general functions on arrangements using graphics hardware; a special case of this result in two dimensions was proved in [16]. In general, the idea of using a family of duals to maintain boundedness of the arrangement can be extended to d dimensions. We defer these more general results to a full version of the paper.

4 Extent Measures

Let $S = \{p_1, \dots, p_n\}$ be a set of points in $\mathbb{R}^d$. We describe streaming algorithms for computing the diameter and width of S for $d \leq 3$ and the smallest enclosing box and ball of S for $d = 2$.

Diameter. In this section we describe a six-pass algorithm for computing the diameter of a set S (the maximum distance between any two points of S) of n

points in $\mathbb{R}^3$. It is well known that the diameter of S is realized by a pair of antipodal points, i.e., there exists a direction u in the positive hemisphere of $\mathbb{S}^2$ such that $\text{diam}(S) = \|\lambda(u, S) - \lambda(-u, S)\| = \|f_H^+(\hat{u}_1, \hat{u}_2) - f_H^-(\hat{u}_1, \hat{u}_2)\|$, where H is the set of planes dual to the points in S . In order to compute $\|f_H^+(x) - f_H^-(x)\|$, we assign the RGB values of the color of a plane p_i^* to be the coordinates of p_i . The first pass computes f_H^+ , so after the pass, the pixel x in the color buffer contains the coordinates of $f_H^+(x)$. We copy this buffer to the texture memory and compute f_H^- in the second pass. We then compute $\|f_H^+(x) - f_H^-(x)\|$ for each pixel. Since the hardware computes these values for $x \in [-1, +1]^2$, we repeat these steps for $R_x(S)$ and $R_y(S)$ as well. Since our algorithm operates in the dual plane, the discretization incurred is in terms of the directions, yielding the following result.

Theorem 1. *Given a point set $S \subset \mathbb{R}^3$, $\alpha > 1$, there is a six-pass $(\alpha, g(\alpha))$ -approximation algorithm for computing the diameter of S , where $g(\alpha) = O(1/\sqrt{\alpha})$.*

Width. Let S be a set of n points in $\mathbb{R}^3$. The *width* of S is the minimum distance between two parallel planes that enclose P between them, i.e., $\text{width}(S) = \min_{u \in \mathbb{S}^2} \omega(u, S)$. The proof of the following lemma is relatively straightforward.

Lemma 2. *Let R_x, R_y be the rotation transforms as described earlier, and let H (resp. H_x, H_y) be the set of planes dual to the points in S (resp. $R_x(S), R_y(S)$). Then*

$$\text{width}(S) = \min_{p \in [-1, +1]^2} \frac{1}{\|(p, 1)\|} \min\{I_H(p), I_{H_x}(p), I_{H_y}(p)\}.$$

This lemma implies that the algorithm for width can be implemented similar to the algorithm for diameter. Consider a set of coplanar points in $\mathbb{R}^3$. No discretized set of directions can yield a good approximation to the width of this set (which is zero). Hence, we can only prove a slightly weaker approximation result, based on knowing a lower bound on the optimal width w^* . We omit the details from this version and conclude the following.

Theorem 2. *Given a point set $S \subset \mathbb{R}^3$, $\alpha > 1$, and $\tilde{w} \leq w^*$, there is a six-pass $(\alpha, g(\alpha, \tilde{w}))$ -approximation algorithm for computing the width of S .*

1-center. The 1-center of a point set S in $\mathbb{R}^2$ is a point $c \in \mathbb{R}^2$ minimizing $\max_{p \in P} d(c, p)$. This is an envelope computation, but in the primal plane. For each point $p \in S$, we render the colored distance cone as described in Section 2. The 1-center is then the point in the upper envelope of the distance cones with the smallest distance value. The center of the smallest enclosing ball will always lie inside $\text{conv}(S)$. The radius of the smallest enclosing ball is at least half the diameter Δ of S . Thus, if we compute the farthest point Voronoi diagram on a grid of cell size $g = \alpha\Delta/2$, the value we obtain is a α -approximation to the radius of the smallest enclosing ball. An approximate diameter computation gives us $\hat{\Delta} \leq 2\Delta$, and thus a grid size of $\alpha\hat{\Delta}/4$ will obtain the desired result.

Theorem 3. *Given a point set S in $\mathbb{R}^2$ and a parameter $\alpha > 1$, there is a two-pass $(\alpha, g(\alpha))$ -approximation algorithm for computing the smallest-area disk enclosing S .*

Smallest bounding box. Let S be a set of points in $\mathbb{R}^2$. A rectangle enclosing S consists of two pairs of parallel lines, lines in each pair orthogonal to the other. For a direction $u \in \mathbb{S}^1$, let $u^\perp$ be the direction normal to u . Then the side lengths of the smallest rectangle whose edges are in directions u and $u^\perp$ that contains S are $W(u) = \omega(u, S)$ and $H(u) = \omega(u^\perp, S)$. Hence, the area of the smallest rectangle containing S is $\min_{u \in \mathbb{S}^1} W(u) \cdot H(u)$. The algorithm to compute the minimum-area two-dimensional bounding box can now be viewed as computing the minimum widths in two orthogonal directions and taking their product. Similarly, we can compute a minimum-perimeter rectangle containing S . Since the algorithm is very similar to computing the width, we omit the details and conclude the following.

Theorem 4. *Given a point set S in $\mathbb{R}^2$, $\alpha > 1$, and a lower bound $\tilde{a}$ on the area of the smallest bounding box, there is a four-pass $(\alpha, g(\alpha, a))$ -approximation algorithm for computing the smallest enclosing bounding box.*

It is not clear how to extend this algorithm to $\mathbb{R}^3$ using a constant number of passes since the set of directions normal to a given direction is $\mathbb{S}^1$. However, by sampling the possible choices of orthogonal directions, we can get a $(1 + \alpha)$ -approximation in $1/\sqrt{\alpha - 1}$ passes. Omitting all the details, we obtain the following.

Theorem 5. *Given point set $S \subset \mathbb{R}^3$, $\alpha > 1$ and lower bound $\tilde{a}$ on the area of the smallest bounding box, there is an $O(1/\sqrt{\alpha - 1})$ -pass $(\alpha, g(\alpha, a))$ -approximation algorithm for computing the smallest bounding box.*

5 Collision Detection

Given two convex polytopes P and Q in $\mathbb{R}^3$, their penetration depth, denoted $PD(P, Q)$ is defined as the length of the shortest translation vector t such that P and $Q + t$ are disjoint. We can specify a placement of Q by fixing a reference point $q \in Q$ and specifying its coordinates. Assume that initially q is at the origin o . Since $M = P \oplus -Q$ is the set of placements of Q at which Q intersects P , $PD(P, Q) = \min_{z \in \partial M} d(o, z)$. For a direction $u \in \mathbb{S}^2$, let $h_M(u)$ be the tangent plane of M normal to direction u . As shown in [1], $PD(P, Q) = \min_{u \in \mathbb{S}^2} d(o, h_M(u))$.

Let A be a convex polytope in $\mathbb{R}^3$ and let V be the set of vertices in A . For a direction $u \in \mathbb{S}^2$, let $g_A(u) = \max_{p \in V} \langle p, \hat{u} \rangle$. It can be verified that the tangent plane of A in direction u is $h_A(u) : \langle \hat{u}, x \rangle = g_A(u)$. Therefore $PD(P, Q) = \min_{u \in \mathbb{S}^2} \frac{g_M(u)}{\|\hat{u}\|}$. The following lemma shows how to compute $h_M(u)$ from $h_P(u)$ and $h_{-Q}(u)$.

Lemma 3. *For any $u \in \mathbb{S}^2$, $g_M(u) = g_P(u) + g_{-Q}(u)$* □

This lemma follows from the fact that for convex P and Q , the point of M extreme in direction u is the sum of the points of P and Q extreme in direction u . Therefore, $PD(P, Q) = \min_{u \in \mathbb{S}^2} g_P(u) + g_{-Q}(u) / \|u\|$.

Hence, we discretize the set of directions in $\mathbb{S}^2$, compute $g_P(u), g_{-Q}(u), (g_P(u) + g_{-Q}(u)) / \|\hat{u}\|$ and compute their minimum. Since g_P and g_{-Q} are upper envelopes of a set of linear functions, they can be computed at a set of directions by graphics hardware in six passes, as described in Section 4. We note here that the above approach can be generalized to compute any inner product-based distance between two non-intersecting convex polytopes in three dimensions. It can also be used to compute the Minkowski sum of polygons in two dimensions.

6 Shape Fitting

We now present hardware-based heuristics for shape analysis problems. These problems are solved in the primal, by computing envelopes of distance functions.

Circle fitting. The *minimum-width annulus* of a point set $P \subset \mathbb{R}^2$ is a pair of concentric disks R_1, R_2 of radii $r_1 > r_2$ such that P lies in the region $R_1 \setminus R_2$ and $r_1 - r_2$ is minimized. Note that the center of the minimum-width annulus could be arbitrarily far away from the point set (for example, the degenerate case of points on a line). Furthermore, when the minimum-width annulus is *thin*, the pixelization induces large errors which cannot be bounded. Therefore, we look at the special case when the annulus is not thin, i.e. $r_1 \geq (1 + \varepsilon)r_2$. For this case, Chan [4] presents a $(1 + \varepsilon)$ approximation algorithm by laying a grid on the pointset, snapping the points to the grid points, and finding the annulus with one of the grid points as the center. This algorithm can be implemented efficiently in hardware as follows: for each point p_i , draw its Euclidean distance cone C_i as described in Section 2. Let $C = \{C_1, C_2, \dots, C_n\}$ be the collection of distance functions. Then the minimum-width annulus can be computed as $\min_{x \in B} I_C(x)$ with center $\arg \min_{x \in B} I_C(x)$. This approach yields a fast streaming $(1 + \varepsilon)$ -approximation algorithm for the minimum-width annulus (and for the minimum-area annulus as well, by using paraboloids instead of cones).

The *best-fit circle* of a set of points $P = \{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^2$ is a circle $C(c, r)$ of radius r centered at c such that the expression $\sum_{p \in P} d^2(p, C)$ is minimized. For a fixed center c , elementary calculus arguments show that the optimal r is given by $r^* = 1/n \sum_{p \in P} d(p, c)$. Let $d_i = \|p_i - c\|$. The cost of the best fit circle of radius r^* centered at c can be shown to be $\sum_{i \leq n} d_i^2 - (1/n)(\sum_{i \leq n} d_i)^2$.

Once again, this function can be represented as an overlay of distance cones, and thus for each grid point, the cost of the optimal circle centered at this grid point can be computed. Unfortunately, this fails to yield an approximation guarantee for the same reasons as above.

Hausdorff distance. Given two point sets $P, Q \subset \mathbb{R}^2$, the *Hausdorff distance* d_H from P to Q is $\max_{p \in P} \min_{q \in Q} d(p, q)$. Once again, we draw distance cones for each point in Q , and compute the lower envelope of this arrangement of surfaces *restricted to points in P* . Now each grid point corresponding to a point of P has a value equal to the distance to the closest point in Q . A maximization

over this set yields the desired result. For this problem, it is easy to see that as for the width, given any lower bound on the Hausdorff distance we can compute a $(\beta, g(\beta))$ -approximation to the Hausdorff distance.

7 Experiments

In this section we describe some implementation specific details, and report empirical results of our algorithms, and compare their performance with software-based approximation algorithms.

Cost bottleneck. The costs of operations can be divided into two types: geometric operations, and fragment operations. Most current graphics cards have a number of *geometry engines* and *raster managers* to handle multiple vertex and fragment operations in parallel. Therefore, we can typically assume that the geometry transformation and each buffer operation takes constant time.

As the number of fragments increases, the rendering time is roughly unchanged till we saturate the rendering capacity (called the fill-rate), at which point performance degrades severely. We now propose a hierarchical method that circumvents the fill limitation by doing refined local searches for the solution.

Hierarchical refinement. One way to alleviate the fill-rate bottleneck is to produce fewer fragments per plane. Instead of sampling the search space with a uniform grid, we instead perform adaptive sampling by constructing a coarse grid, computing the solution value for each grid point and then recursively refining candidate points. The advantage of using adaptive refinement is that not all the grid cells need to be refined to a high resolution. However, the local search performed by this selective refinement could fail to find an approximate solution with the guarantee implied by this higher resolution. In our experiments, we will compare the results obtained from this approach with those obtained by software-based methods.

Empirical results. In this section we report on the performance of our algorithms. All our algorithms were implemented in C++ and OpenGL, and run on a 2.4GHz Pentium IV Linux PC with an ATI Radeon 9700 graphics card and 512 MB Memory. Our experiments were run on three types of inputs: (i) randomly generated convex shapes [9] (ii) large geometric models of various objects, available at <http://www.cc.gatech.edu/graphmodels/> and (iii) randomly generated input using `rbox` (a component of `qhull`). In all our algorithms we use hierarchical refinement (with depth two) to achieve more accurate solutions.

Penetration depth. We compare our implementation of penetration depth (called **HwPD**) with our own implementation of an exact algorithm (called **SwPD**) based on Minkowski sums which exhibits quadratic complexity and with DEEP [14], which to the best of our knowledge is the only other implementation for penetration depth. We used the convex polytopes available at [9], as well as random polytopes found by computing the convex hull of points on random ellipsoids as inputs to test our code. The performance of the algorithms on the input set is presented

in Table 1. **HwPD** always outperforms **SwPD** in running time, in some cases by over three orders of magnitude. With regard to **DEEP**, the situation is less clear. **DEEP** performs significant preprocessing on its input, so a single number is not representative of the running times for either program. Hence, we report both preprocessing times and query times (for our code, preprocessing time is merely reading the input). We note that **DEEP** crashed on some of our inputs; we mark those entries with an asterisk.

Table 1. Comparison of running times for penetration depth (in secs.). On the last three datasets, we stopped **SwPD** after it ran for over 25 minutes. Asterisks mark inputs for which **DEEP** crashed.

Polygon		HwPD			DEEP			SwPD	
Size	Size	Preproc.	Time	Depth	Preproc.	Time	Depth	Time	Depth
500	500	0	0.04	1.278747	0.15	0	1.29432	27.69	1.289027
750	750	0	0.08	1.053032	0.25	0	1.07359	117.13	1.071013
789	1001	0.01	0.067	1.349714	*	*	*	148.87	1.364840
789	5001	0.01	0.17	1.360394	*	*	*	-	-
5001	4000	0.02	0.30	1.362190	*	*	*	-	-
10000	5000	0.04	0.55	1.359534	3.28	0	1.4443	-	-

2D minimum width annulus. We compute an annulus by laying a $1/\varepsilon^2 \times 1/\varepsilon^2$ grid on the pointset, snapping the points to the grid, and then using the hardware to find the nearest/furthest neighbour of each grid point. The same algorithm can be implemented in software. We compare our implementation (called **HAnnWidth**) with the software implementation, called **SAnnWidth**. The input point sets to the programs were synthetically generated using **rbox**: R-Circle-r refers to a set of points with minimum width annulus r and is generated by sampling points from a circle and introducing small perturbations. See Table 2.

Table 2. Comparison of running time and approximation for 2D-min width annulus

Error:	$\epsilon^2 = 0.002$	HAnnWidth		SAnnWidth	
Dataset	size	Time	Width	Time	Width
R-Circle-0.1	(1,000)	0.36	0.099882	0.53	0.099789
R-Circle-0.2	(1,000)	0.35	0.199764	0.42	0.199442
R-Circle-0.1	(2,000)	0.66	0.099882	0.63	0.099816
R-Circle-0.1	(5,000)	1.58	0.099882	26.44	0.099999
R-Circle-0.1	(10,000)	3.12	0.099882	0.93	0.099999

3D width. We compare our implementation of width (called **HWidth**) with the code of Duncan *et al.* [5] (**DGRWidth**). Algorithm **DGRWidth** reduces the computation of the width to $O(1/\epsilon)$ linear programs. It then tries certain pruning heuristics to reduce the number of linear programs solved in practice. The performance of both the algorithms on a set of real graphical models is presented in Table 3: column four gives the $(1 + \epsilon)$ -approximate value of the width computed by the two algorithms for the ϵ given in the second column (this ϵ value

dictates the window size required by our algorithm, as explained previously, and the number of linear programs solved by DGRWidth). HWidth always outperforms DGRWidth in running time, in some cases by more than a factor of five.

Table 3. Comparison of running time and approximation quality for 3D-width.

		Error	HWidth		DGRWidth	
Dataset	size	ϵ	Time	Width	Time	Width
Club	(16,864)	0.250	0.45	0.300694	0.77	0.312883
Bunny	(35,947)	0.060	0.95	1.276196	2.70	1.29231
Phone	(83,034)	0.125	2.55	0.686938	6.17	0.697306
Human	(254,721)	0.180	6.53	0.375069	18.91	0.374423
Dragon	(437,645)	0.075	10.88	0.813487	39.34	0.803875
Blade	(882,954)	0.090	23.45	0.715578	66.71	0.726137

3D diameter. We compare our implementation (HDiam) with the approximation algorithm of Malandain and Boissonnat [18] (MBDiam), and Har-Peled [10] (PDiam). PDiam maintains a hierarchical decomposition of the point set, and iteratively throws away pairs that are not candidate for the diameter until an approximate distance is achieved by a pair of points. MBDiam is a further improvement on PDiam. Table 4 reports the timing and approximation comparisons for two error measures for graphical models. Although our running times in this case are worse than the software implementations, they are comparable even for very large inputs, illustrating the generality of our approach.

Table 4. Comparison of running time and approximation quality for 3D-diameter.

Error: $\epsilon = 0.015$		HDiam		MBDiam		PDiam	
Dataset	size	Time	Diam	Time	Diam	Time	Diam
Club	(16,864)	0.023	2.326992	0.0	2.32462	0.00	2.32462
Bunny	(35,947)	0.045	2.549351	0.75	2.54772	0.03	2.54772
Phone	(83,034)	0.11	2.416497	0.01	2.4115	0.07	2.4115
Human	(254,721)	0.32	2.020594	3.5	2.01984	0.04	2.01938
Dragon	(437,645)	0.55	2.063075	17.27	2.05843	0.21	2.05715
Blade	(882,954)	1.10	2.246725	0.1	2.23939	0.22	2.22407

References

1. AGARWAL, P., GUIBAS, L., HAR-PELED, S., RABINOVITCH, A., AND SHARIR, M. Penetration depth of two convex polytopes in 3d. *Nordic J. Comput.* 7, 3 (2000), 227–240.
2. AGARWAL, P. K., HAR-PELED, S., AND VARADARAJAN, K. Approximating extent measures of points. Submitted for publication, 2002.
3. AGARWAL, P. K., AND SHARIR, M. Efficient algorithms for geometric optimization. *ACM Comput. Surv.* 30 (1998), 412–458.

4. CHAN, T. M. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *Proc. 16th Annu. Sympos. on Comp. Geom.* (2000), pp. 300–309.
5. DUNCAN, C., GOODRICH, M., AND RAMOS, E. Efficient approximation and optimization algorithms for computational metrology. In *ACM-SIAM Symp. Discrete Algo.* (1997), pp. 121–130.
6. FEIGENBAUM, J., KANNAN, S., AND ZHANG, J. Computing diameter in the streaming and sliding-window models. DIMACS Working Group on Streaming Data Analysis II, 2003.
7. FOURNIER, A., AND FUSSELL, D. On the power of the frame buffer. *ACM Transactions on Graphics* (1988), 103–128.
8. GUHA, S., KRISHNAN, S., MUNAGALA, K., AND VENKATASUBRAMANIAN, S. The power of a two-sided depth test and its application to CSG rendering and depth extraction. Tech. rep., AT&T, 2002.
9. HAR-PELED, S. <http://valis.cs.uiuc.edu/sariel/research/papers/99/nav/nav.html>.
10. HAR-PELED, S. A practical approach for computing the diameter of a point-set. In *Proc. 17th Annu. Symp. on Comp. Geom.* (2001), pp. 177–186.
11. HERSBERGER, J., AND SURI, S. Convex hulls and related problems in data streams. In *SIGMOD-DIMACS MPDS Workshop* (2003).
12. HOFF III, K. E., KEYSER, J., LIN, M., MANOCHA, D., AND CULVER, T. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics 33*, Annual Conference Series (1999), 277–286.
13. INDYK, P. Stream-based geometric algorithms. In *SIGMOD-DIMACS MPDS Workshop* (2003).
14. KIM, Y. J., LIN, M. C., AND MANOCHA, D. Fast penetration depth estimation between polyhedral models using hierarchical refinement. In *6th Intl. Workshop on Algo. Founda. of Robotics* (2002).
15. KORN, F., MUTHUKRISHNAN, S., AND SRIVASTAVA, D. Reverse nearest neighbour aggregates over data streams. In *Proc. 28th Conf. VLDB* (2002).
16. KRISHNAN, S., MUSTAFA, N., AND VENKATASUBRAMANIAN, S. Hardware-assisted computation of depth contours. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms* (2002), pp. 558–567.
17. MAJHI, J., JANARDAN, R., SMID, M., AND SCHWERDT, J. Multi-criteria geometric optimization problems in layered manufacturing. In *Proc. 14th Annu. Symp. on Comp. Geom.* (1998), pp. 19–28.
18. MALANDAIN, G., AND BOISSONNAT, J.-D. Computing the diameter of a point set. In *Discrete Geometry for Computer Imagery* (Bordeaux, France, 2002), A. Braquelaire, J.-O. Lachaud, and A. Vialard, Eds., vol. 2301 of *LNCS*, Springer.
19. MUSTAFA, N., KOUTSOFIOS, E., KRISHNAN, S., AND VENKATASUBRAMANIAN, S. Hardware assisted view dependent map simplification. In *Proc. 17th Annu. Symp. on Comp. Geom.* (2001), pp. 50–59.
20. MUTHUKRISHNAN, S. Data streams: Algorithms and applications. Tech. rep., Rutgers University, 2003.
21. WOO, M., NEIDER, J., DAVIS, T., AND SHREINER, D. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*, 3 ed. Addison-Wesley, 1999.

An Efficient Implementation of a Quasi-polynomial Algorithm for Generating Hypergraph Transversals*

E. Boros¹, K. Elbassioni¹, V. Gurvich¹, and Leonid Khachiyan²

¹ RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854-8003;
`{boros,elbassio,gurvich}@rutcor.rutgers.edu`

² Department of Computer Science, Rutgers University, 110 Frelinghuysen Road,
Piscataway NJ 08854-8003; `leonid@cs.rutgers.edu`

Abstract. Given a finite set V , and a hypergraph $\mathcal{H} \subseteq 2^V$, the hypergraph transversal problem calls for enumerating all minimal hitting sets (transversals) for $\mathcal{H}$. This problem plays an important role in practical applications as many other problems were shown to be polynomially equivalent to it. Fredman and Khachiyan (1996) gave an incremental quasi-polynomial time algorithm for solving the hypergraph transversal problem [9]. In this paper, we present an efficient implementation of this algorithm. While we show that our implementation achieves the same bound on the running time as in [9], practical experience with this implementation shows that it can be substantially faster. We also show that a slight modification of the algorithm in [9] can be used to give a stronger bound on the running time.

1 Introduction

Let V be a finite set of cardinality $|V| = n$. For a hypergraph $\mathcal{H} \subseteq 2^V$, let us denote by $\mathcal{I}(\mathcal{H})$ the family of its *maximal independent* sets, i.e. maximal subsets of V not containing any hyperedge of $\mathcal{H}$. The complement of a maximal independent subset is called a *minimal transversal* of $\mathcal{H}$ (i.e. minimal subset of V intersecting all hyperedges of $\mathcal{H}$). The collection $\mathcal{H}^d$ of minimal transversals is also called the *dual* or *transversal* hypergraph for $\mathcal{H}$. The *hypergraph transversal problem* is the problem of generating all transversals of a given hypergraph. This problem has important applications in combinatorics [14], artificial intelligence [8], game theory [11,12], reliability theory [7], database theory [6,8,10], integer programming [3], learning theory [1], and data mining [2,5,6].

The *theoretically* best known algorithm for solving the hypergraph transversal problem is due to Fredman and Khachiyan [9] and works by performing $|\mathcal{H}^d| + 1$ calls to the following problem, known as *hypergraph dualization*:

* This research was supported by the National Science Foundation (Grant IIS-0118635), and by the Office of Naval Research (Grant N00014-92-J-1375). The third author is also grateful for the partial support by DIMACS, the National Science Foundation's Center for Discrete Mathematics and Theoretical Computer Science.

DUAL($\mathcal{H}, \mathcal{X}$): *Given a complete list of all hyperedges of $\mathcal{H}$, and a set of minimal transversals $\mathcal{X} \subseteq \mathcal{H}^d$, either prove that $\mathcal{X} = \mathcal{H}^d$, or find a new transversal $X \in \mathcal{H}^d \setminus \mathcal{X}$.*

Two recursive algorithms were proposed in [9] to solve the hypergraph dualization problem. These algorithms have incremental quasi-polynomial time complexities of $\text{poly}(n) + m^{O(\log^2 m)}$ and $\text{poly}(n) + m^{o(\log m)}$ respectively, where $m = |\mathcal{H}| + |\mathcal{X}|$. Even though the second algorithm is theoretically more efficient, the first algorithm is much simpler in terms of its implementation overhead, making it more attractive for practical applications. In fact, as we have found out experimentally, in many cases the most critical parts of the dualization procedure, in terms of execution time, are operations performed in each recursive call, rather than the total number of recursive calls. With respect to this measure, the first algorithm is more efficient due to its simplicity. For that reason, we present in this paper an implementation of the first algorithm in [9], which is efficient with respect to the time per recursive call. We further show that this efficiency in implementation does not come at the cost of increasing the worst-case running time substantially.

Rather than considering the hypergraph dualization problem, we shall consider, in fact, the more general problem of dualization on boxes introduced in [3]. In this latter problem, we are given an integral box $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$, where $\mathcal{C}_i$ is a finite set of consecutive integers, and a subset $\mathcal{A} \subseteq \mathcal{C}$. Denote by $\mathcal{A}^+ = \{x \in \mathcal{C} \mid x \geq a, \text{ for some } a \in \mathcal{A}\}$ and $\mathcal{A}^- = \{x \in \mathcal{C} \mid x \leq a, \text{ for some } a \in \mathcal{A}\}$, the ideal and filter generated by $\mathcal{A}$. Any element in $\mathcal{C} \setminus \mathcal{A}^+$ is called *independent of $\mathcal{A}$* , and we let $\mathcal{I}(\mathcal{A})$ denote the set of all maximal independent elements for $\mathcal{A}$. Given $\mathcal{A} \subseteq \mathcal{C}$ and a subset $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A})$ of maximal independent elements of $\mathcal{A}$, problem **DUAL**($\mathcal{C}, \mathcal{A}, \mathcal{B}$) calls for generating a new element $x \in \mathcal{I}(\mathcal{A}) \setminus \mathcal{B}$, or proving that there is no such element. By performing $|\mathcal{I}(\mathcal{A})| + 1$ calls to problem **DUAL**($\mathcal{C}, \mathcal{A}, \mathcal{B}$), we can solve the following problem

GEN($\mathcal{C}, \mathcal{A}$): *Given an integral box $\mathcal{C}$, and a subset of vectors $\mathcal{A} \subseteq \mathcal{C}$, generate all maximal independent elements of $\mathcal{A}$.*

Problem **GEN**($\mathcal{C}, \mathcal{A}$) has several interesting applications in integer programming and data mining, see [3,4,5] and the references therein. Extensions of the two hypergraph transversal algorithms mentioned above to solve problem **DUAL**($\mathcal{C}, \mathcal{A}, \mathcal{B}$) were given in [3]. In this paper, we give an implementation of the first dualization algorithm in [3], which achieves efficiency in two directions:

- Re-use of the recursion tree: dualization-based techniques generate all maximal independent elements of a given subset $\mathcal{A} \subseteq \mathcal{C}$ by usually performing $|\mathcal{I}(\mathcal{A})| + 1$ calls to problem **DUAL**($\mathcal{C}, \mathcal{A}, \mathcal{B}$), thus building a new recursion tree for each call. However, as it will be illustrated, it is more efficient to use the same recursion tree to generate all the elements of $\mathcal{I}(\mathcal{A})$, since the recursion trees required to generate many elements may be nearly identical.
- Efficient implementation at each recursion tree node: Straight forward implementation of the algorithm in [3] requires $O(n|\mathcal{A}| + n|\mathcal{B}|)$ time per recursive

call. However, this can be improved to $O(n|A| + |B| + n \log(|B|))$ by maintaining a binary search tree on the elements of B , and using randomization. Since $|B|$ is usually much larger than $|A|$, this gives a significant improvement. Several heuristics are also used to improve the running time. For instance, we use random sampling to find the branching variable and its value, required to divide the problem at the current recursion node. We also estimate the numbers of elements of A and B that are active at the current node, and only actually compute these active elements when their numbers drop by a certain factor. As our experiments indicate, such heuristics can be very effective in practically improving the running time of the algorithm.

The rest of this paper is organized as follows. In section 2 we introduce some basic terminology used throughout the paper, and briefly outline the Fredman-Khachiyan algorithm (or more precisely, its generalization to boxes). Section 3 describes the data structure used in our implementation, and Section 4 presents the algorithm. In Section 5, we show that the new version of the algorithm has, on the expected, the same quasi-polynomial bound on the running time as that of [3], and we also show how to get a slightly stronger bound on the running time. Section 6 briefly outlines our preliminary experimental findings with the new implementation for generating hypergraph transversals. Finally, we draw some conclusions in Section 7.

2 Terminology and Outline of the Algorithm

Throughout the paper, we assume that we are given an integer box $C^* = C_1^* \times \dots \times C_n^*$, where $C_i^* = [l_i^* : u_i^*]$, and $l_i^* \leq u_i^*$, are integers, and a subset $A^* \subseteq C^*$ of vectors for which it is required to generate all maximal independent elements. The algorithm of [3], considered in this paper, solves problem $\text{DUAL}(C, A, B)$, by decomposing it into a number of smaller subproblems and solving each of them recursively. The input to each such subproblem is a sub-box C of the original box C^* and two subsets $A \subseteq A^*$ and $B \subseteq B^*$ of integral vectors, where $B^* \subseteq \mathcal{I}(A^*)$ denotes the subfamily of maximal independent elements that the algorithm has generated so far. Note that, by definition, the following condition holds for the original problem and all subsequent subproblems:

$$a \not\leq b, \text{ for all } a \in A, b \in B. \quad (1)$$

Given an element $a \in A$ ($b \in B$), we say that a coordinate $i \in [n] \stackrel{\text{def}}{=} \{1, \dots, n\}$ is *essential* for a (respectively, b), in the box $C = [l_1 : u_1] \times \dots \times [l_n : u_n]$, if $a_i > l_i$ (respectively, if $b_i < u_i$). Let us denote by $\text{Ess}(x)$ the set of essential coordinates of an element $x \in A \cup B$. Finally, given a sub-box $C \subseteq C^*$, and two subsets $A \subseteq A^*$ and $B \subseteq B^*$, we shall say that B is *dual to* A in C if $A^+ \cup B^- \supseteq C$.

A key lemma, on which the algorithm in [3] is based, is that either (i) there is an element $x \in A \cup B$ with at most $1/\epsilon$ essential coordinates, where $\epsilon \stackrel{\text{def}}{=} 1/(1 + \log m)$ and $m \stackrel{\text{def}}{=} |A| + |B|$, or (ii) one can easily find a new maximal

independent element $z \in \mathcal{C}$, by picking each element z_i independently at random from $\{l_i, u_i\}$ for $i = 1, \dots, n$; see subroutine `Random solution( $\cdot, \cdot, \cdot$ )` in the next section. In case (i), one can decompose the problem into two strictly smaller subproblems as follows. Assume, without loss of generality, that $x \in \mathcal{A}$ has at most $1/\epsilon$ essential coordinates. Then, by (1), there is an $i \in [n]$ such that $|\{b \in \mathcal{B} : b_i < x_i\}| \geq \epsilon|\mathcal{B}|$. This allows us to decompose the original problem into two subproblems $\text{DUAL}(\mathcal{C}', \mathcal{A}, \mathcal{B}')$ and $\text{DUAL}(\mathcal{C}'', \mathcal{A}'', \mathcal{B})$, where $\mathcal{C}' = \mathcal{C}_1 \times \dots \times \mathcal{C}_{i-1} \times [x_i : u_i] \times \mathcal{C}_{i+1} \times \dots \times \mathcal{C}_n$, $\mathcal{B}' = \mathcal{B} \cap \mathcal{C}^+$, $\mathcal{C}'' = \mathcal{C}_1 \times \dots \times \mathcal{C}_{i-1} \times [l_i : x_i - 1] \times \mathcal{C}_{i+1} \times \dots \times \mathcal{C}_n$, and $\mathcal{A}'' = \mathcal{A} \cap \mathcal{C}^-$. This way, the algorithm is guaranteed to reduce the cardinality of one of the sets $\mathcal{A}$ or $\mathcal{B}$ by a factor of at least $1 - \epsilon$ at each recursive step. For efficiency reasons, we do two modifications to this basic approach. First, we use sampling to estimate the sizes of the sets $\mathcal{B}'$, $\mathcal{A}''$ (see subroutine `Est( $\cdot, \cdot$ )` below). Second, once we have determined the new sub-boxes $\mathcal{C}'$, $\mathcal{C}''$ above, we do not compute the *active* families $\mathcal{B}'$ and $\mathcal{A}''$ at each recursion step (this is called the Cleanup step in the next section). Instead, we perform the cleanup step only when the number of vectors reduces by a certain factor f , say $1/2$, for two reasons: First, this improves the running time since the elimination of vectors is done less frequently. Second, the expected total memory required by all the nodes of the path from the root of the recursion tree to a leaf is at most $O(nm + m/(1 - f))$, which is linear in m for constant f .

3 The Data Structure

We use the following data structures in our implementation:

- Two arrays of vectors, A and B containing the elements of $\mathcal{A}^*$ and $\mathcal{B}^*$ respectively.
- Two (dynamic) arrays of indices, $\text{index}(\mathcal{A})$ and $\text{index}(\mathcal{B})$, containing the indices of vectors from $\mathcal{A}^*$ and $\mathcal{B}^*$ (i.e. containing pointers to elements of the arrays A and B), that appear in the current subproblem. These arrays are used to enable sampling from the sets $\mathcal{A}$ and $\mathcal{B}$, and also to keep track of which vectors are currently active, i.e. intersect the current box.
- A balanced binary search tree $\mathbf{T}(\mathcal{B}^*)$, built on the elements of $\mathcal{B}^*$ using lexicographic ordering. Each node of the tree contains an index of an element in the array B . This way, checking whether a given vector $x \in \mathcal{C}$ belongs to $\mathcal{B}^*$ or not, takes only $O(n \log |\mathcal{B}^*|)$ time.

4 The Algorithm

In the sequel, we let $m = |\mathcal{A}| + |\mathcal{B}|$ and $\epsilon = 1/(1 + \log m)$. We assume further that operations of the form $\mathcal{A}'' \leftarrow \mathcal{A}$ and $\mathcal{B}' \leftarrow \mathcal{B}$ are actually performed on the index arrays $\text{index}(A)$, $\text{index}(B)$, so that they only take $O(m)$ rather than $O(nm)$ time. We use the following subroutines in our implementation:

- $\max_{\mathcal{A}}(z)$. It takes as input a vector $z \notin \mathcal{A}^+$ and returns a maximal vector z^* in $(\mathcal{C}^* \cap \{z\}^+) \setminus \mathcal{A}^+$. This can be done in $O(n|\mathcal{A}|)$ by initializing $c(a) = |\{i \in$

- $[n] : a_i > z_i\}$ for all $a \in \mathcal{A}$, and repeating, for $i = 1, \dots, n$, the following two steps: (i) $z_i^* \leftarrow \min(u_i^*, \min\{a_i - 1 : a \in \mathcal{A}, c(a) = 1 \text{ and } a_i > z_i\})$ (where we assume $\min(\emptyset) = \infty$); (ii) $c(a) \leftarrow c(a) - 1$ for each $a \in \mathcal{A}$ such that $z_i < a_i \leq z_i^*$.
- Exhaustive duality($\mathcal{C}, \mathcal{A}, \mathcal{B}$). Assuming $|\mathcal{A}||\mathcal{B}| \leq 1$, check duality in $O(n(|\mathcal{A}^*| + \log |\mathcal{B}|))$ as follows: First, if $|\mathcal{A}| = |\mathcal{B}| = 1$ then find an $i \in [n]$ such that $a_i > b_i$, where $\mathcal{A} = \{a\}$ and $\mathcal{B} = \{b\}$. (Such a coordinate is guaranteed to exist by (1).) If there is a $j \neq i$ such that $b_j < u_j$ then return $\max_{\mathcal{A}^*}(u_1, \dots, u_{i-1}, b_i, u_{i+1}, \dots, u_n)$. If there is a $j \neq i$ such that $a_j > l_j$ then return $(u_1, \dots, u_{j-1}, a_j - 1, u_{j+1}, \dots, u_n)$. If $b_i < a_i - 1$ then return $(u_1, \dots, u_{i-1}, a_i - 1, u_{i+1}, \dots, u_n)$. Otherwise return *FALSE* (meaning that $\mathcal{A}$ and $\mathcal{B}$ are dual in $\mathcal{C}$). Second, if $|\mathcal{A}| = 0$ then let $z = \max_{\mathcal{A}^*}(u)$, and return either *FALSE* or z depending on whether $z \in \mathcal{B}^*$ or not (this check can be done in $O(n \log |\mathcal{B}^*|)$ using the search tree $\mathbf{T}(\mathcal{B}^*)$). Finally, if $|\mathcal{B}| = 0$ then return either *FALSE* or $z = \max_{\mathcal{A}^*}(l)$ depending on whether $l \in \mathcal{A}^+$ or not (this check requires $O(n|\mathcal{A}|)$ time).
 - Random solution($\mathcal{C}, \mathcal{A}^*, \mathcal{B}$). Repeat the following for $k = 1, \dots, t_1$ times, where t_1 is a constant (say 10): Find a random point $z^k \in \mathcal{C}$, by picking each coordinate z_i^k randomly from $\{l_i, u_i\}$, $i = 1, \dots, n$. Let $(z^k)^* \leftarrow \max_{\mathcal{A}^*}(z^k)$. If $(z^k)^* \notin \mathcal{B}^*$ then return $(z^k)^*$. If $\{(z^1)^*, \dots, (z^{t_1})^*\} \subseteq \mathcal{B}^*$ then return *FALSE*. This step takes $O(n(|\mathcal{A}^*| + \log |\mathcal{B}^*|))$ time, and is used to check whether $\mathcal{A}^+ \cup \mathcal{B}^-$ covers a large portion of $\mathcal{C}$.
 - Count estimation. For a subset $\mathcal{X} \subseteq \mathcal{A}$ (or $\mathcal{X} \subseteq \mathcal{B}$), use sampling to estimate the number $\text{Est}(\mathcal{X}, \mathcal{C})$ of elements of $\mathcal{X} \subseteq \mathcal{A}$ (or $\mathcal{X} \subseteq \mathcal{B}$) that are active with respect to the current box $\mathcal{C}$, i.e. the elements of the set $\mathcal{X}' \stackrel{\text{def}}{=} \{a \in \mathcal{X} \mid a^+ \cap \mathcal{C} \neq \emptyset\}$ ($\mathcal{X}' \stackrel{\text{def}}{=} \{b \in \mathcal{X} \mid b^- \cap \mathcal{C} \neq \emptyset\}$). This can be done as follows. For $t_2 = O(\log(|\mathcal{A}| + |\mathcal{B}|)/\epsilon)$, pick elements $x^1, \dots, x^{t_2} \in \mathcal{A}$ at random, and let the random variable $Y = \frac{|\mathcal{A}|}{t_2} * |\{x^i \in \mathcal{X}' : i = 1, \dots, t_2\}|$. Repeat this step independently for a total of $t_3 = O(\log(|\mathcal{A}| + |\mathcal{B}|))$ times to obtain t_3 estimates $Y^1, \dots, Y^{t_3}$, and let $\text{Est}(\mathcal{X}, \mathcal{C}) = \min\{Y^1, \dots, Y^{t_3}\}$. This step requires $O(n \log^3 m)$ time.¹
 - Cleanup($\mathcal{A}, \mathcal{C}$) (Cleanup($\mathcal{B}, \mathcal{C}$)). Set $\mathcal{A}' \leftarrow \{a \in \mathcal{A} \mid a^+ \cap \mathcal{C} \neq \emptyset\}$ (respectively, $\mathcal{B}' \leftarrow \{b \in \mathcal{B} \mid b^- \cap \mathcal{C} \neq \emptyset\}$), and return $\mathcal{A}'$ (respectively, $\mathcal{B}'$). This step takes $O(n|\mathcal{A}|)$ (respectively, $O(n|\mathcal{B}|)$).

Now, we describe the implementation of procedure GEN-DUAL($\mathcal{A}, \mathcal{B}, \mathcal{C}$) which is called initially using $\mathcal{C} \leftarrow \mathcal{C}^*$, $\mathcal{A} \leftarrow \mathcal{A}^*$ and $\mathcal{B} \leftarrow \emptyset$. At the return of this call, $\mathcal{B}$ is extended by the elements in $\mathcal{I}(\mathcal{A}^*)$. Below we assume that $f \in (0, 1)$ is a constant.

¹ Note that these sample sizes were chosen to theoretically get a guarantee on the expected running time of the algorithm. However, as our experiments indicate, smaller (usually constant) sample sizes are enough to provide practically good performance.

Procedure GEN-DUAL($\mathcal{C}, \mathcal{A}, \mathcal{B}$):Input: A box $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$ and subsets $\mathcal{A} \subseteq \mathcal{A}^* \subseteq \mathcal{C}$, and $\mathcal{B} \subseteq \mathcal{I}(\mathcal{A}^*)$.Output: A subset $\mathcal{N} \subseteq \mathcal{I}(\mathcal{A}^*) \setminus \mathcal{B}$.

1. $\mathcal{N} \leftarrow \emptyset$.
2. While $|\mathcal{A}||\mathcal{B}| \leq 1$
 - 2.1. $z \leftarrow \text{Exhaustive duality}(\mathcal{C}, \mathcal{A}, \mathcal{B})$.
 - 2.2. If $z = \text{FALSE}$ then return($\mathcal{N}$).
 - 2.3. $\mathcal{B} \leftarrow \mathcal{B} \cup \{z\}$, $\mathcal{N} \leftarrow \mathcal{N} \cup \{z\}$.
- end while
3. $z \leftarrow \text{Random Solution}(\mathcal{C}, \mathcal{A}^*, \mathcal{B})$.
4. While ($z \neq \text{FALSE}$) do
 - 4.1. $\mathcal{B} \leftarrow \mathcal{B} \cup \{z\}$, $\mathcal{N} \leftarrow \mathcal{N} \cup \{z\}$.
 - 4.2. $z \leftarrow \text{Random Solution}(\mathcal{C}, \mathcal{A}^*, \mathcal{B})$.
- end while
5. $x^* \leftarrow \text{argmin}\{|\text{Ess}(y)| : y \in (\mathcal{A} \cap \mathcal{C}^-) \cup (\mathcal{B} \cap \mathcal{C}^+)\}$.
6. If $x^* \in \mathcal{A}$ then
 - 6.1. $i \leftarrow \text{argmax}\{\text{Est}(\{b \in \mathcal{B} : b_j < x_j^*\}, \mathcal{C}) : j \in \text{Ess}(x^*)\}$.
 - 6.2. $\mathcal{C}' = \mathcal{C}_1 \times \cdots \times \mathcal{C}_{i-1} \times [x_i^* : u_i] \times \mathcal{C}_{i+1} \times \cdots \times \mathcal{C}_n$.
 - 6.3. If $\text{Est}(\mathcal{B}, \mathcal{C}') \leq f * |\mathcal{B}|$ then
 - 6.3.1. $\mathcal{B}' \leftarrow \text{Cleanup}(\mathcal{B}, \mathcal{C}')$.
 - 6.4. else
 - 6.4.1. $\mathcal{B}' \leftarrow \mathcal{B}$.
 - 6.5. $\mathcal{N}' \leftarrow \text{GEN-DUAL}(\mathcal{C}', \mathcal{A}, \mathcal{B}')$.
 - 6.6. $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}'$, $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{N}'$.
 - 6.7. $\mathcal{C}'' = \mathcal{C}_1 \times \cdots \times \mathcal{C}_{i-1} \times [l_i : x_i^* - 1] \times \mathcal{C}_{i+1} \times \cdots \times \mathcal{C}_n$.
 - 6.8. If $\text{Est}(\mathcal{A}, \mathcal{C}'') \leq f * |\mathcal{A}|$ then
 - 6.8.1. $\mathcal{A}'' \leftarrow \text{Cleanup}(\mathcal{A}, \mathcal{C}'')$.
 - 6.9. else
 - 6.9.1. $\mathcal{A}'' \leftarrow \mathcal{A}$.
 - 6.10. $\mathcal{N}'' \leftarrow \text{GEN-DUAL}(\mathcal{C}'', \mathcal{A}'', \mathcal{B})$.
 - 6.11. $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}''$, $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{N}''$.
7. else
 - 7.1-7.11. Symmetric versions for Steps 6.1-6.11 above (details omitted).
- end if
8. Return ($\mathcal{N}$).

5 Analysis of the Expected Running Time

Let $C(v)$ be the expected number of recursive calls on a subproblem $\text{GEN-DUAL}(\mathcal{C}, \mathcal{A}, \mathcal{B})$ of volume $v \stackrel{\text{def}}{=} |\mathcal{A}||\mathcal{B}|$. Consider a particular recursive call of the algorithm and let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be the current inputs to this call. Let x^* be the element with minimum number of essential coordinates found in Step 5, and assume without loss of generality that $x^* \in \mathcal{A}$. As mentioned before, we assume also that the factor f used in Steps 6.3 and 6.8 is $1/2$. For $i = 1, \dots, n$, let $\mathcal{B}_i \stackrel{\text{def}}{=} \{b \in \mathcal{B} : b_i < x_i^*\}$, and denote by $\overline{\mathcal{B}} = \mathcal{B} \cap \mathcal{C}^+$ and $\overline{\mathcal{B}}_i = \mathcal{B}_i \cap \mathcal{C}^+$ the

subsets of $\mathcal{B}$ and $\mathcal{B}_i$ that are active with respect to the current box $\mathcal{C}$. In this section, we show that our implementation has, with high probability, almost the same quasi-polynomial bound on the running time as the algorithm of [3].

Lemma 1. *Suppose that $k \in \text{Ess}(x^*)$ satisfies $|\overline{\mathcal{B}}_k| \geq \epsilon |\overline{\mathcal{B}}|$. Let $i \in [n]$ be the coordinate obtained in Step 6.1 of the algorithm, and $v = |\mathcal{A}||\mathcal{B}|$. Then*

$$\Pr \left[|\overline{\mathcal{B}}_i| \geq \epsilon \frac{|\overline{\mathcal{B}}|}{4} \right] \geq 1 - \frac{1}{\epsilon v}. \quad (2)$$

Proof. For $j = 1, \dots, n$, let $Y_j \stackrel{\text{def}}{=} \text{Est}(\mathcal{B}_j, \mathcal{C})$. Then the random variable $X_j \stackrel{\text{def}}{=} t_2 Y_j / |\mathcal{B}|$ is Binomially distributed with parameters t_2 and $|\overline{\mathcal{B}}_j|/|\mathcal{B}|$, and thus by Chernoff Bound

$$\Pr[Y_j < \frac{\mathbb{E}[Y_j]}{2}] < e^{-\mathbb{E}[X_j]/8}, \quad \text{for } j = 1, \dots, n.$$

In particular, for $j = k$, we get $\Pr[Y_k < \epsilon \frac{|\overline{\mathcal{B}}|}{2}] < e^{-\mathbb{E}[X_k]/8}$ since $\mathbb{E}[Y_k] \geq \epsilon |\overline{\mathcal{B}}|$. Note that, since $\text{Est}(\mathcal{B}, \mathcal{C})$ is the minimum over t_3 independent trials, it follows by Markov Inequality that if $|\overline{\mathcal{B}}| < |\mathcal{B}|/4$, then $\Pr[\text{Est}(\mathcal{B}, \mathcal{C}) < |\mathcal{B}|/2] > 1 - 2^{-t_3}$, and the cleanup step will be performed with high probability. On the other hand, if $|\overline{\mathcal{B}}| \geq |\mathcal{B}|/4$ then $\mathbb{E}[X_k] = t_2 |\overline{\mathcal{B}}_k|/|\mathcal{B}| \geq \epsilon t_2/4$. Thus, it follows that $\Pr[Y_k < \epsilon \frac{|\overline{\mathcal{B}}|}{2}] < e^{-\epsilon t_2/32} + 2^{-t_3}$. Moreover, for any $j \in \text{Ess}(x^*)$ for which $|\overline{\mathcal{B}}_j|/|\overline{\mathcal{B}}| < \epsilon/4$, we have $\Pr[Y_j \geq \epsilon \frac{|\overline{\mathcal{B}}|}{2}] < 2^{-t_3}$. Consequently,

$$\begin{aligned} \Pr \left[Y_k \geq \frac{\epsilon |\overline{\mathcal{B}}|}{2} \text{ and } Y_j < \frac{\epsilon |\overline{\mathcal{B}}|}{2} \text{ for all } j \in \text{Ess}(x^*) \text{ such that } \frac{|\overline{\mathcal{B}}_j|}{|\overline{\mathcal{B}}|} < \frac{\epsilon}{4} \right] \\ > 1 - 2^{-t_3} |\text{Ess}(x^*)| - e^{-\epsilon t_2/32} \geq 1 - \frac{1}{\epsilon v}, \end{aligned}$$

where the last inequality follows by our selection of t_2 and t_3 . Since, in Step 6.1, we select the index $i \in [n]$ maximizing Y_i , we have $Y_i \geq Y_k$ and thus, with probability at least $1 - 1/(\epsilon v)$, we have $|\overline{\mathcal{B}}_i|/|\overline{\mathcal{B}}| \geq \epsilon/4$. $\square$

Lemma 2. *The expected number of recursive calls until a new maximal independent element is output, or procedure GEN-DUAL($\mathcal{C}, \mathcal{A}, \mathcal{B}$) terminates is $nm^{O(\log^2 m)}$.*

Proof. For a node N of the recursion tree, denote by $\overline{\mathcal{A}} = \overline{\mathcal{A}}(N)$, $\overline{\mathcal{B}} = \overline{\mathcal{B}}(N)$ the subsets of $\mathcal{A}$ and $\mathcal{B}$ intersecting the box specified by node N , and let $v(N) = |\overline{\mathcal{A}}(N)||\overline{\mathcal{B}}(N)|$. Now consider the node N at which the latest maximal independent element was generated. If $s \stackrel{\text{def}}{=} \sum_{a \in \overline{\mathcal{A}}(N)} (1/2)^{|\text{Ess}(a)|} + \sum_{b \in \overline{\mathcal{B}}(N)} (1/2)^{|\text{Ess}(b)|} < 1/2$, then the probability that the point $z \in \mathcal{C}$, picked randomly in Steps 3 or 4.2 of the procedure, belongs to $\overline{\mathcal{A}}(N)^+ \cup \overline{\mathcal{B}}(N)^-$ is at most $\sigma_1 \stackrel{\text{def}}{=} (1/2)^{t_1}$. Thus, in this case, with probability at least $1 - \sigma_1$, we find a new maximal independent element. Assume therefore that $s \geq 1/2$, let x^* be

the element with $|\text{Ess}(x^*)| \leq 1/\epsilon$ found in Step 5, and assume without loss of generality that $x^* \in \mathcal{A}$. Then, by (1), there exists a coordinate $k \in \text{Ess}(x^*)$ such that $|\overline{\mathcal{B}}_k| \geq \epsilon|\overline{\mathcal{B}}|$. By Lemma 1, with probability at least $\sigma_2 \stackrel{\text{def}}{=} 1 - 1/(\epsilon v)$, we can reduce the volume of one of the subproblems, of the current problem, by a factor of at least $1 - \epsilon/4$. Thus for the expected number of recursive calls at node N , we get the following recurrence

$$C(v) \leq \frac{1}{\sigma_2} \left[1 + C(v-1) + C\left(\left(1 - \frac{\epsilon}{4}\right)v\right) \right], \quad (3)$$

where $v = v(N)$. This recurrence gives $C(v) \leq v^{O(\log^2 v)}$. Now consider the path $N_0 = N, N_1, \dots, N_r$ from node N to the root of the recursion tree N_r . Since a large number of new maximal independent elements may have been added at node N (and of course to all its ancestors in the tree), recurrence (3) may no longer hold at nodes $N_1, \dots, N_r$. However, since we count the number of recursive calls from the time of the last generation that happened at node N , each node N_i , that has N_{i-1} as a right child in the tree, does not contribute to this number. Furthermore, the number of recursive calls resulting from the right child of each node N_i , that has N_{i-1} as a left child, is at most $C(v(N_r))$. Since the number of such nodes does not exceed the depth of the tree, which is at most nm , the expected total number of recursive calls is at most $nmC(v(N_r))$ and the lemma follows. $\square$

We show further that, if $|\mathcal{B}| \gg |\mathcal{A}|$, i.e. if the output size is much bigger than the input size, then the number of recursive calls required for termination, after the last dual element is generated by $\text{GEN-DUAL}(\mathcal{A}, \mathcal{B}, \mathcal{C})$, is $nm^{o(\log^2 m)}$.

Lemma 3. *Suppose that $\mathcal{A}$ and $\mathcal{B}$ are dual in $\mathcal{C}$, then the expected number of recursive calls until $\text{GEN-DUAL}(\mathcal{C}, \mathcal{A}, \mathcal{B})$ terminates is $nm^{O(\delta \log m)}$, where $m = |\mathcal{A}| + |\mathcal{B}|$ and $\delta = \min\{\log \alpha, \frac{\log(\beta/\alpha)}{c(\alpha, \beta/\alpha)}, \frac{\log(\alpha/\beta)}{c(\beta, \alpha/\beta)}\} + 1$, $\alpha = |\mathcal{A}|$, $\beta = |\mathcal{B}|$, and $c = c(a, b)$ is the unique positive root of the equation*

$$2^c \left(a^{c/\log b} - 1 \right) = 1. \quad (4)$$

Proof. Let $r = \min\{|\text{Ess}(y)| : y \in \mathcal{A} \cup \mathcal{B}\}$, $p = \left(1 + \left(\frac{\beta}{\alpha}\right)^{\frac{1}{r-1}}\right)^{-1}$, and let $z \in \mathcal{C}$ be a random element obtained by picking each coordinate independently with $\Pr[z_i = l_i] = p$ and $\Pr[z_i = u_i] = 1 - p$. Then the probability that $z \in \mathcal{A}^+ \cup \mathcal{B}^-$ is at most $\sum_{a \in \mathcal{A}} (1-p)^{|\text{Ess}(a)|} + \sum_{b \in \mathcal{B}} p^{|\text{Ess}(b)|} \leq \alpha(1-p)^r + \beta p^r = \beta p^{r-1}$. Since $\mathcal{A}$ and $\mathcal{B}$ are dual in $\mathcal{C}$, it follows that $\beta p^{r-1} \geq 1$, and thus

$$r - 1 \leq \frac{\log \beta}{\log(1 + (\beta/\alpha)^{\frac{1}{r-1}})}. \quad (5)$$

The maximum value that r can achieve is when both sides of (5) are equal, i.e. r is bounded by the root r' of the equation $\beta^{1/(r'-1)} = 1 + (\beta/\alpha)^{1/(r'-1)}$. If $\alpha = \beta$, then $r' = \log \alpha + 1$. If $\beta > \alpha$, then letting $(\beta/\alpha)^{1/(r'-1)} = 2^c$, we get

$$r' = 1 + \frac{\log(\beta/\alpha)}{c(\alpha, \beta/\alpha)}, \quad (6)$$

where $c(\cdot, \cdot)$ is as defined in (4). The case for $\alpha > \beta$ is similar and the lemma follows from (1) and Lemma 2. $\square$

Note that, if β is much larger than α , then the root r' in (6) is approximately

$$r' \sim 1 + \frac{\log(\beta/\alpha)}{\log(\log(\beta/\alpha)/\log \alpha)}$$

and thus the expected running time of procedure GEN-DUAL($\mathcal{A}, \mathcal{B}, \mathcal{C}$), from the time of last output till termination, is $nm^{o(\log^2 m)}$. In fact, one can use Lemma 2 together with the method of conditional expectations to obtain an incremental deterministic algorithm for solving problem GEN($\mathcal{C}, \mathcal{A}$), whose delay between any two successive outputs is of the order given by Lemma 2.

6 Experimental Results

We performed a number of experiments to evaluate our implementation. Five types of hypergraphs were used in the experiments:

- Random (denoted henceforth by $R(n, \alpha, d)$): this is a hypergraph with α hyperedges, each of which is picked randomly by first selecting its size k uniformly from $[2 : d]$ and then randomly selecting k elements of $[n]$ (in fact, in some experiments, we fix $k = d$ for all hyperedges).
- Matching ($M(n)$): this is a graph on n vertices (n is even) with $n/2$ edges forming an induced matching.
- Matching Dual ($MD(n)$): this is just $M(n)^d$, the transversal hypergraph of $M(n)$. In particular, it has $2^{n/2}$ hyperedges on n vertices.
- Threshold graph ($TH(n)$): this is a graph on n vertices numbered from 1 to n (where n is even), with edge set $\{\{i, j\} : 1 \leq i < j \leq n, j \text{ is even}\}$ (i.e., for $j = 2, 4, \dots, n$, there is an edge between i and j for all $i < j$). The reason we are interested in such kind of graphs is that they are known to have both a small number of edges (namely, $n^2/4$) and a small number of transversals (namely, $n/2 + 1$ for even n).
- Self-dualized threshold graph ($SDTH(n)$): this is a self-dual hypergraph $\mathcal{H}$ on n vertices obtained from the threshold graph and its dual $TH(n-2)$, $TH(n-2)^d \subseteq 2^{[n-2]}$ as follows:

$$\mathcal{H} = \{\{n-1, n\}\} \bigcup \{\{n-1\} \cup H \mid H \in TH(n-2)\} \bigcup \{\{n\} \cup H \mid H \in TH(n-2)^d\}.$$

This gives a family of hypergraphs with polynomially bounded input and output sizes $|SDTH(n)| = |SDTH(n)^d| = (n-2)^2/4 + n/2 + 1$.

- Self-dualized Fano-plane product ($SDFP(n)$): this is constructed by starting with the hypergraph $\mathcal{H}_0 = \{\{1, 2, 3\}, \{1, 5, 6\}, \{1, 7, 4\}, \{2, 4, 5\}, \{2, 6, 7\}, \{3, 4, 6\}, \{3, 5, 7\}\}$ (which represents the set of lines in a Fano plane and is self-dual), taking $k = (n-2)/7$ disjoint copies $\mathcal{H}_1, \dots, \mathcal{H}_k$ of $\mathcal{H}_0$, and letting $\mathcal{H} = \mathcal{H}_1 \cup \dots \cup \mathcal{H}_k$. The dual hypergraph $\mathcal{H}^d$ is just the hypergraph of all 7^k unions obtained by taking one hyperedge from each of the hypergraphs

$\mathcal{H}_1, \dots, \mathcal{H}_k$. Finally, we define the hypergraph $SDFP(k)$ to be the hypergraph of $1 + 7k + 7^k$ hyperedges on n vertices, obtained by self-dualizing $\mathcal{H}$ as we did for threshold graphs.

Table 1. Performance of the algorithm for different classes of hypergraphs. Numbers below parameters indicate the total CPU time, in seconds, taken to generate all transversals.

$R(n, \alpha, d)$ $2 \leq d \leq n - 1$	$n = 30$			$n = 50$			$n = 60$		
	$\alpha = 275$	$\alpha = 213$	$\alpha = 114$	$\alpha = 507$	$\alpha = 441$	$\alpha = 342$	$\alpha = 731$	$\alpha = 594$	$\alpha = 520$
	0.1	0.3	3.1	43.3	165.6	1746.8	322.2	2220.4	13329.5
$M(n)$	$n = 20$	$n = 24$	$n = 28$	$n = 30$	$n = 32$	$n = 34$	$n = 36$	$n = 38$	$n = 40$
	0.3	1.4	7.1	17.8	33.9	80.9	177.5	418.2	813.1
$MD(n)$	$n = 20$	$n = 24$	$n = 28$	$n = 30$	$n = 32$	$n = 34$	$n = 36$	$n = 38$	$n = 40$
	0.15	1.3	13.3	42.2	132.7	421.0	1330.3	4377.3	14010.5
$TH(n)$	$n = 40$	$n = 60$	$n = 80$	$n = 100$	$n = 120$	$n = 140$	$n = 160$	$n = 180$	$n = 200$
	0.4	1.9	6.0	18.4	40.2	78.2	142.2	232.5	365.0
$SDTH(n)$	$n = 42$	$n = 62$	$n = 82$	$n = 102$	$n = 122$	$n = 142$	$n = 162$	$n = 182$	$n = 202$
	0.9	5.9	23.2	104.0	388.3	1164.2	2634.0	4820.6	8720.0
$SDFP(n)$	$k = 16$	$n = 23$		$n = 30$		$n = 37$			
	0.1	4.8		198.1		11885.1			

The experiments were performed on a *Pentium 4* processor with 2.2 GHz of speed and 512M bytes of memory. Table 1 summarizes our results for several instances of the different classes of hypergraphs listed above. In the table, we show the total CPU time, in seconds, required to generate all transversals for the specified hypergraphs, with the specified parameters. For random hypergraphs, the time reported is the average over 30 experiments. The average sizes of the transversal hypergraphs, corresponding to the random hypergraphs in Table 1 are (from left to right): 150, 450, $5.7 \cdot 10^3$, $1.7 \cdot 10^4$, $6.4 \cdot 10^4$, $4.7 \cdot 10^5$, $7.5 \cdot 10^4$, $4.7 \cdot 10^5$, and $1.7 \cdot 10^6$, respectively. The output sizes for the other classes of hypergraphs can be computed using the formulas given above. For instance, for $SDTH(n)$, with $n = 202$ vertices, the number of hyperedges is $\alpha = 10102$. For random hypergraphs, we only show results for $n \leq 60$ vertices. For larger numbers of vertices, the number of transversals becomes very large (although the delay between successive transversals is still acceptable).

We also performed some experiments to compare different implementations of the algorithm and to study the effect of increasing the number of vertices and the number of hyperedges on the performance. In particular, Figure 1 shows the effect of rebuilding the tree each time a transversal is generated on the output rate. From this figure we see that the average time per transversal is almost constant if we do not rebuild the tree. In Figure 2, we show that the randomized implementation of the algorithm offers substantial improvement over the deterministic one. Figures 3 and 4, respectively, show how the average CPU time/transversal changes as the number of vertices n and the number of hyperedges α are increased. The plots show that the average CPU time/transversal does not increase more than linearly with increasing α or n .

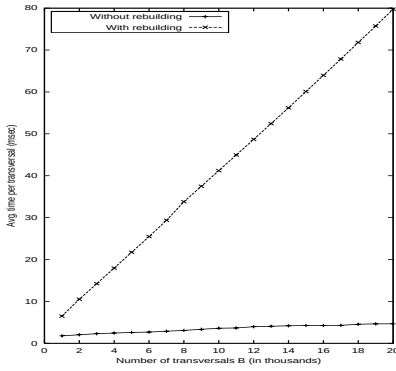


Fig. 1. Effect of rebuilding the recursion tree. Each plot shows the average CPU time (in milli-seconds) per generated transversal versus the number of transversals, for hypergraphs of type $R(30, 100, 5)$.

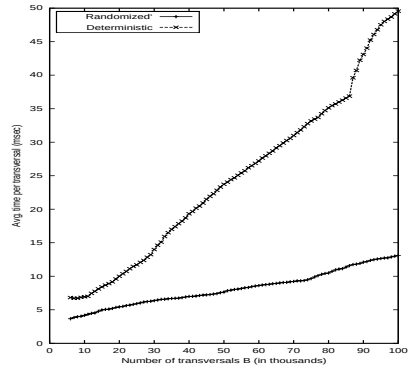


Fig. 2. Comparing deterministic versus randomized implementations. Each plot shows the average CPU time/transversal versus the number of transversals, for hypergraphs of type $R(50, 100, 10)$.

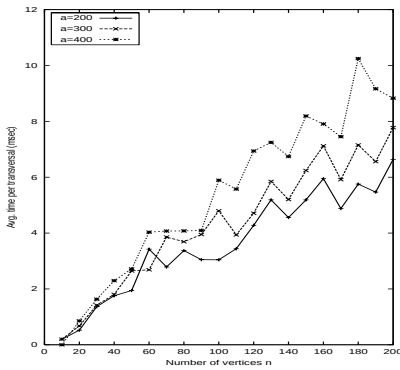


Fig. 3. Average CPU time/transversal versus the number of vertices n for random hypergraphs $R(n, a, d)$, where $d = n/4$, and $a = 200, 300, 400$.

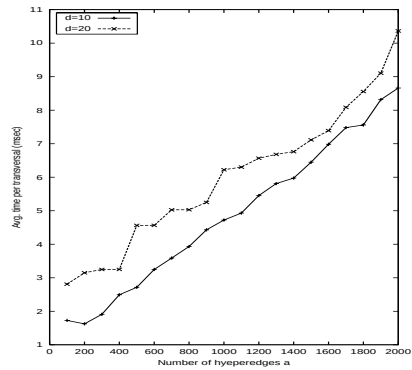


Fig. 4. Average CPU time/transversal versus the number of hyperedges a for random hypergraphs $R(50, a, d)$, for $d = 10, 20$.

7 Conclusion

We have presented an efficient implementation of an algorithm for generating maximal independent elements for a family of vectors in an integer box. Experiments show that this implementation performs well in practice. We are not aware of any experimental evaluation of algorithms for generating hypergraph transversals except for [13] in which a heuristic for solving this problem was described and experimentally evaluated. However, the results in [13] show the performance for relatively small instances which are easy cases for our implemen-

tation. On the other hand, the method described in this paper can handle much larger instances due to the fact that it scales nicely with the size of the problem. In particular, our code can produce, in a few hours, millions of transversals even for hypergraphs with hundreds of vertices and thousands of hyperedges. Furthermore, the experiments also indicate that the delay per transversal scales almost linearly with the number of vertices and number of hyperedges.

Acknowledgements. We thank the referees for the helpful remarks.

References

1. M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge Univ. Press, 1992.
2. R. Agrawal, T. Imielinski and A. Swami, Mining associations between sets of items in massive databases, *Proc. 1993 ACM-SIGMOD Int. Conf.*, pp. 207–216.
3. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K. Makino, Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities, *SIAM Journal on Computing*, **31** (5) (2002) pp. 1624–1643.
4. E. Boros, K. Elbassioni, V. Gurvich and L. Khachiyan, Generating Dual-Bounded Hypergraphs, *Optimization Methods and Software*, (OMS) **17** (5), Part I (2002), pp. 749–781.
5. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K. Makino, An intersection inequality for discrete distributions and related generation problems, to appear in *ICALP 2003*.
6. E. Boros, V. Gurvich, L. Khachiyan and K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets, in *19th Int. Symp. on Theoretical Aspects of Computer Science, (STACS)*, March 2002, LNCS 2285, pp. 133–141.
7. C. J. Colbourn, *The combinatorics of network reliability*, Oxford Univ. Press, 1987.
8. T. Eiter and G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24 (1995) pp. 1278–1304.
9. M. L. Fredman and L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, *Journal of Algorithms*, 21 (1996) pp. 618–628.
10. D. Gunopulos, R. Khardon, H. Mannila and H. Toivonen, Data mining, hypergraph transversals and machine learning, in *Proc. 16th ACM-PODS Conf.*, (1997) pp. 209–216.
11. V. Gurvich, To theory of multistep games, *USSR Comput. Math. and Math Phys.* **13-6** (1973), pp. 1485–1500.
12. V. Gurvich, Nash-solvability of games in pure strategies, *USSR Comput. Math. and Math. Phys.*, **15** (1975), pp. 357–371.
13. D. J. Kavvadias and E. C. Stavropoulos, Evaluation of an algorithm for the transversal hypergraph problem, in *Proc. 3rd Workshop on Algorithm Engineering (WAE'99)*, LNCS 1668, pp. 72–84, 1999.
14. R. C. Read, Every one a winner, or how to avoid isomorphism when cataloging combinatorial configurations, *Annals of Disc. Math.* 2 (1978) pp. 107–120.

Experiments on Graph Clustering Algorithms^{*}

Ulrik Brandes¹, Marco Gaertler², and Dorothea Wagner²

¹ University of Passau, Department of Mathematics & Computer Science,
94030 Passau, Germany. brandes@algo.fmi.uni-passau.de

² University of Karlsruhe, Faculty of Informatics, 76128 Karlsruhe, Germany.
{dwagner,gaertler}@ira.uka.de

Abstract. A promising approach to graph clustering is based on the intuitive notion of intra-cluster density vs. inter-cluster sparsity. While both formalizations and algorithms focusing on particular aspects of this rather vague concept have been proposed no conclusive argument on their appropriateness has been given.

As a first step towards understanding the consequences of particular conceptions, we conducted an experimental evaluation of graph clustering approaches. By combining proven techniques from graph partitioning and geometric clustering, we also introduce a new approach that compares favorably.

1 Introduction

Clustering is an important issue in the analysis and exploration of data. There is a wide area of applications as e.g. data mining, VLSI design, computer graphics and gene analysis. See also [1] and [2] for an overview. Roughly speaking, clustering consists in discovering natural groups of similar elements in data sets. An interesting and important variant of data clustering is graph clustering. On one hand, similarity is often expressed by a graph. On the other hand, there is a growing interest in network analysis in general.

A natural notion of graph clustering is the separation of sparsely connected dense subgraphs from each other. Several formalizations have been proposed. However, the understanding of current algorithms and indices is still rather intuitive. As a first step towards understanding the consequences of particular conceptions, we concentrate on indices and algorithms that focus on the relation between the number of intra-cluster and inter-cluster edges.

In [3] some indices measuring the quality of a graph clustering are discussed. Conductance, an index concentrating on the intra-cluster edges is introduced and a clustering algorithm that repeatedly separates the graph is presented. A graph clustering algorithm incorporating the idea of performing a random walk on the graph to identify the more densely connected subgraphs is presented in [4] and the index *performance* is considered to measure the quality of a graph

^{*} This work was partially supported by the DFG under grant BR 2158/1-1 and WA 654/13-1 and EU under grant IST-2001-33555 COSIN.

clustering. The idea of random walks is also used in [5] but only for clustering geometric data. Obviously, there is a close connection between graph clustering and the classical graph problem minimum cut. A purely graph-theoretic approach using this connection more or less directly is the recursive minimum cut approach presented in [6]. Other more advanced partition techniques involve spectral information as in [3,7,8,9].

It is not precisely known how well indices formalizing the relation between the number of intra-cluster and inter-cluster edges measure the quality of a graph clustering. Moreover, there exists no conclusive evaluation of algorithms that focus on such indices. In this paper, we give a summary of those indices and conduct an experimental evaluation of graph clustering approaches. The already known algorithms under comparison are the iterative conductance cut algorithm presented in [3] and the Markov clustering approach from [4]. By combining proven techniques from graph partitioning and geometric clustering, we also introduce a new approach that compares favorably with respect to flexibility and running time.

In Section 2 the notation used throughout the paper is introduced and clustering indices considered in the experimental study are presented. Section 3 gives a detailed description of the three algorithms considered. The graph generators used for the experimental evaluation are described in Section 4.1 and the results of the evaluation are summarized in Section 4.3.

2 Indices for Graph Clustering

Throughout this paper we assume that $G = (V, E)$ is a connected, undirected graph. Let $|V| =: n$, $|E| =: m$ and $\mathcal{C} = (C_1, \dots, C_k)$ a partition of V . We call $\mathcal{C}$ a *clustering* of G and the C_i *clusters*; $\mathcal{C}$ is called *trivial* if either $k = 1$, or all clusters C_i contain only one element. In the following, we often identify a cluster C_i with the induced subgraph of G , i.e. the graph $G[C_i] := (C_i, E(C_i))$, where $E(C_i) := \{\{v, w\} \in E : v, w \in C_i\}$. Then $E(\mathcal{C}) := \bigcup_{i=1}^k E(C_i)$ is the set of *intra-cluster edges* and $E \setminus E(\mathcal{C})$ the set of *inter-cluster edges*. The number of intra-cluster edges is denoted by $m(\mathcal{C})$ and the number of inter-cluster edges by $\overline{m}(\mathcal{C})$. A clustering $\mathcal{C} = (C, V \setminus C)$ is also called a *cut* of G and $\overline{m}(\mathcal{C})$ the *size* of the cut. A cut with minimum size is called a *mincut*.

2.1 Coverage

The *coverage*($\mathcal{C}$) of a graph clustering $\mathcal{C}$ is the fraction of intra-cluster edges within the complete set of edges, i.e.

$$\text{coverage}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} = \frac{m(\mathcal{C})}{m(\mathcal{C}) + \overline{m}(\mathcal{C})}.$$

Intuitively, the larger the value of *coverage*($\mathcal{C}$) the better the quality of a clustering $\mathcal{C}$. Notice that a mincut has maximum coverage and in this sense would be an “optimal” clustering. However, in general a mincut is not considered

to be a good clustering of a graph. Therefore, additional constraints on the number of clusters or the size of the clusters seem to be reasonable. While a mincut can be computed in polynomial time, constructing a clustering with a fixed number k , $k \geq 3$ of clusters is NP-hard [10], as well as finding a mincut satisfying certain size constraints on the clusters [11].

2.2 Performance

The *performance*($\mathcal{C}$) of a clustering $\mathcal{C}$ counts the number of “correctly interpreted pairs of nodes” in a graph. More precisely, it is the fraction of intra-cluster edges together with non-adjacent pairs of nodes in different clusters within the set of all pairs of nodes, i.e.

$$\text{performance}(\mathcal{C}) := \frac{m(\mathcal{C}) + \sum_{\{v,w\} \notin E, v \in C_i, w \in C_j, i \neq j} 1}{\frac{1}{2}n(n-1)}.$$

Calculating the performance of a clustering according to this formula would be quadratic in the number of nodes. Especially, if the performance has to be computed for a sequence of clusterings of the same graph, it might be more efficient to count the number of “errors” instead (Equation (1)). Maximizing the performance is reducible to graph partitioning which is NP-hard [12].

$$1 - \text{performance}(\mathcal{C}) = \frac{2m(1 - 2\text{coverage}(\mathcal{C})) + \sum_{i=1}^k |C_i|(|C_i| - 1)}{n(n-1)} \quad (1)$$

2.3 Intra- and Inter-cluster Conductance

The *conductance of a cut* compares the size of the cut and the number of edges in either of the two induced subgraphs. Then the *conductance* $\phi(G)$ of a graph G is the minimum conductance value over all cuts of G . For a clustering $\mathcal{C} = (C_1, \dots, C_k)$ of a graph G , the *intra-cluster conductance* $\alpha(\mathcal{C})$ is the minimum conductance value over all induced subgraphs $G[C_i]$, while the *inter-cluster conductance* $\delta(\mathcal{C})$ is the maximum conductance value over all induced cuts $(C_i, V \setminus C_i)$. For a formal definition of the different notions of conductance, let us first consider a cut $\mathcal{C} = (C, V \setminus C)$ of G and define *conductance* $\phi(C)$ and $\phi(G)$ as follows.

$$\phi(C) := \begin{cases} 1, & C \in \{\emptyset, V\} \\ 0, & C \notin \{\emptyset, V\} \text{ and } \overline{m}(\mathcal{C}) = 0 \\ \frac{\overline{m}(\mathcal{C})}{\min(\sum_{v \in C} \deg v, \sum_{v \in V \setminus C} \deg v)}, & \text{otherwise} \end{cases}$$

$$\phi(G) := \min_{C \subseteq V} \phi(C)$$

Then a cut has small conductance if its size is small relative to the density of either side of the cut. Such a cut can be considered as a bottleneck. Minimizing the conductance over all cuts of a graph and finding the according cut is

NP-hard [10], but can be approximated with poly-logarithmic approximation guarantee in general, and constant approximation guarantee for special cases, [9] and [8]. Based on the notion of conductance, we can now define intra-cluster conductance $\alpha(\mathcal{C})$ and inter-cluster conductance $\delta(\mathcal{C})$.

$$\alpha(\mathcal{C}) := \min_{i \in \{1, \dots, k\}} \phi(G[C_i]) \quad \text{and} \quad \delta(\mathcal{C}) := 1 - \max_{i \in \{1, \dots, k\}} \phi(C_i)$$

In a clustering with small intra-cluster conductance there is supposed to be at least one cluster containing a bottleneck, i.e. the clustering is possibly too coarse in this case. On the other hand, a clustering with small inter-cluster conductance is supposed to contain at least one cluster that has relatively strong connections outside, i.e. the clustering is possibly too fine. To see that a clustering with maximum intra-cluster conductance can be found in polynomial time, consider first $m = 0$. Then $\alpha(\mathcal{C}) = 0$ for every non-trivial clustering $\mathcal{C}$, since it contains at least one cluster C_j with $\phi(G[C_j]) = 0$. If $m \neq 0$, consider an edge $\{u, v\} \in E$ and the clustering $\mathcal{C}$ with $C_1 = \{u, v\}$, and $|C_i| = 1$ for $i \geq 2$. Then $\alpha(\mathcal{C}) = 1$, which is maximum.

So, intra-cluster conductance has some artificial behavior for clusterings with many small clusters. This justifies the restriction to clusterings satisfying certain additional constraints on the size or number of clusters. However, under these constraints maximizing intra-cluster conductance becomes an NP-hard problem. Finding a clustering with maximum inter-cluster conductance is NP-hard as well, because it is at least as hard as finding a cut with minimum conductance.

3 Graph Clustering Algorithms

Two graph clustering algorithms that are assumed to perform well with respect to the indices described in the previous section are outlined. The first one iteratively emphasises intra-cluster over inter-cluster connectivity and the second one repeatedly refines an initial partition based on intra-cluster conductance. While both essentially operate locally, we also propose another, more global method. In all three cases, the asymptotic worst-case running time of the algorithms depend on certain parameters given as input. However, notice that for meaningful choices of these parameters, the time complexity of the new algorithm GMC is better than for the other two.

All three algorithms employ the *normalized adjacency matrix* of G , i.e., $M(G) = D(G)^{-1}A(G)$ where $A(G)$ is the adjacency matrix and $D(G)$ the diagonal matrix of vertex degrees.

3.1 Markov Clustering (MCL)

The key intuition behind *Markov Clustering* (MCL) [4, p. 6] is that a “random walk that visits a dense cluster will likely not leave the cluster until many of its vertices have been visited.” Rather than actually simulating random walks, MCL iteratively modifies a matrix of transition probabilities. Starting from $M =$

$M(G)$ (which corresponds to random walks of length at most one), the following two operations are iteratively applied:

- *expansion*, in which M is taken to the power $e \in \mathbb{N}_{>1}$ thus simulating e steps of a random walk with the current transition matrix (Algorithm 1, Step 1)
- *inflation*, in which M is re-normalized after taking every entry to its r th power, $r \in \mathbb{R}^+$. (Algorithm 1, Steps 2–4)

Note that for $r > 1$, inflation emphasizes the heterogeneity of probabilities within a row, while for $r < 1$, homogeneity is emphasized. The iteration is halted upon reaching a recurrent state or a fixpoint. A recurrent state of period $k \in \mathbb{N}$ is a matrix that is invariant under k expansions and inflations, and a fixpoint is a recurrent state of period 1. It is argued that MCL is most likely to end up in a fixpoint [4]. The clustering is induced by connected components of the graph underlying the final matrix. Pseudo-code for MCL is given in Algorithm 1. Except for the stop criterion, MCL is deterministic, and its complexity is dominated by the expansion operation which essentially consists of matrix multiplication.

Algorithm 1: Markov Clustering (MCL)

Input: $G = (V, E)$, expansion parameter e , inflation parameter r

$M \leftarrow M(G)$

while M is not fixpoint **do**

1	$M \leftarrow M^e$				
2	forall $u \in V$ do				
3	<table border="0" style="border-collapse: collapse;"> <tr> <td style="padding-right: 10px; vertical-align: top;">3</td> <td style="border-left: 1px solid black; padding-left: 10px; vertical-align: top;"> forall $v \in V$ do $M_{uv} \leftarrow M_{uv}^r$ </td> </tr> <tr> <td style="padding-right: 10px; vertical-align: top;">4</td> <td style="border-left: 1px solid black; padding-left: 10px; vertical-align: top;"> forall $v \in V$ do $M_{uv} \leftarrow \frac{M_{uv}}{\sum_{w \in V} M_{uw}}$ </td> </tr> </table>	3	forall $v \in V$ do $M_{uv} \leftarrow M_{uv}^r$	4	forall $v \in V$ do $M_{uv} \leftarrow \frac{M_{uv}}{\sum_{w \in V} M_{uw}}$
3	forall $v \in V$ do $M_{uv} \leftarrow M_{uv}^r$				
4	forall $v \in V$ do $M_{uv} \leftarrow \frac{M_{uv}}{\sum_{w \in V} M_{uw}}$				

$H \leftarrow$ graph induced by non-zero entries of M

$\mathcal{C} \leftarrow$ clustering induced by connected components of H

3.2 Iterative Conductance Cutting (ICC)

The basis of *Iterative Conductance Cutting* (ICC) [3] is to iteratively split clusters using minimum conductance cuts. Finding a cut with minimum conductance is NP-hard, therefore the following poly-logarithmic approximation algorithm is used. Consider the vertex ordering implied by an eigenvector to the second largest eigenvalue of $M(G)$. Among all cuts that split this ordering into two parts, one of minimum conductance is chosen. Splitting of a cluster ends when the approximation value of the conductance exceeds an input threshold α^* first. Pseudo-code for ICC is given in Algorithm 2. Except for the eigenvector computations, ICC is deterministic. While the overall running time depends on the number of iterations, the running time of the conductance cut approximation is dominated by the eigenvector computation which needs to be performed in each iteration.

Algorithm 2: Iterative Conductance Cutting (ICC)

Input: $G = (V, E)$, conductance threshold $0 < \alpha^* < 1$

$\mathcal{C} \leftarrow \{V\}$

while there is a $C \in \mathcal{C}$ with $\phi(G[C]) < \alpha^*$ **do**

$x \leftarrow$ eigenvector of $M(G[C])$ associated with second largest eigenvalue

$\mathcal{S} \leftarrow \left\{ S \subset C : \max_{v \in S} \{x_v\} < \min_{w \in C \setminus S} \{x_w\} \right\}$

$C' \leftarrow \arg \min_{S \in \mathcal{S}} \{\phi(S)\}$

$\mathcal{C} \leftarrow (\mathcal{C} \setminus \{C\}) \cup \{C', C \setminus C'\}$

3.3 Geometric MST Clustering (GMC)

Geometric MST Clustering (GMC), is a new graph clustering algorithm combining spectral partitioning with a geometric clustering technique. A geometric embedding of G is constructed from d distinct eigenvectors $x_1, \dots, x_d$ of $M(G)$ associated with the largest eigenvalues less than 1. The edges of G are then weighted by a distance function induced by the embedding, and a minimum spanning tree (MST) of the weighted graph is determined. A MST T implies a sequence of clusterings as follows: For a threshold value τ let $F(T, \tau)$ be the forest induced by all edges of T with weight at most τ . For each threshold τ , the connected components of $F(T, \tau)$ induce a clustering. Note that there are at most $n - 1$ thresholds resulting in different forests. Because of the following nice property of the resulting clustering, we denote it with $\mathcal{C}(\tau)$. The proof of Lemma 1 is omitted. See [13].

Lemma 1. *The clustering induced by the connected components of $F(T, \tau)$ is independent of the particular MST T .*

Among the $\mathcal{C}(\tau)$ we choose one optimizing some measure of quality. Potential measures of quality are, e.g., the indices defined in Section 2, or combinations thereof. This genericity allows to target different properties of a clustering. Pseudo-code for GMC is given in Algorithm 3. Except for the eigenvector computations, GMC is deterministic. Note that, different from ICC, they form a preprocessing step, with their number bounded by a (typically small) input parameter. Assuming that the quality measure can be computed fast, the asymptotic time and space complexity of the main algorithm is dominated by the MST computation. GMC combines two proven concepts from geometric clustering and graph partitioning. The idea of using a MST that way has been considered before [14]. However, to our knowledge the MST decomposition was only used for geometric data before, not for graphs. In our case, general graphs without additional geometric information are considered. Instead, spectral graph theory is used [15] to obtain a geometric embedding that already incorporates insight about dense subgraphs. This induces a canonical distance on the edges which is taken for the MST computation.

Algorithm 3: Geometric MST Clustering (GMC)

Input: $G = (V, E)$, embedding dimension d , clustering valuation *quality*

$(1, \lambda_1, \dots, \lambda_d) \leftarrow d + 1$ largest eigenvalues of $M(G)$

$d' \leftarrow \max \{i : 1 \leq i \leq d, \lambda_i > 0\}$

$x^{(1)}, \dots, x^{(d')} \leftarrow$ eigenvectors of $M(G)$ associated with $\lambda_1, \dots, \lambda_{d'}$

forall $e = (u, v) \in E$ **do** $w(e) \leftarrow \sum_{i=1}^{d'} \left| x_u^{(i)} - x_v^{(i)} \right|$

$T \leftarrow$ MST of G with respect to w

$C \leftarrow \mathcal{C}(\tau)$ for which *quality*($\mathcal{C}(\tau)$) is maximum over all $\tau \in \{w(e) : e \in T\}$

4 Experimental Evaluation

First we describe the general model used to generate appropriate instances for the experimental evaluation. Then we present the experiments and discuss the results of the evaluation.

4.1 Random Uniform Clustered Graphs

We use a random partition generator $\mathcal{P}(n, s, v)$ that determines a partition $(P_1, \dots, P_k)$ of $\{1, \dots, n\}$ with $|P_i|$ being a normal random variable with expected value s and standard deviation $\frac{s}{v}$. Note that k depends on the choice of n, s and v , and that the last element $|P_k|$ of $\mathcal{P}(n, s, v)$ is possibly significantly smaller than the others. Given a partition $\mathcal{P}(n, s, v)$ and probabilities p_{in} and p_{out} , a uniformly random clustered graph $(G, \mathcal{C})$ is generated by inserting intra-cluster edges with probability p_{in} and inter-cluster edges with probability p_{out} ¹. For a clustered graph $(G, \mathcal{C})$ generated that way, the expected values of m , $m(\mathcal{C})$ and $\overline{m}(\mathcal{C})$ can be determined. We obtain

$$\mathbb{E}[\overline{m}(\mathcal{C})] = \frac{p_{\text{out}}}{2} (n(n-s)) \quad \text{and} \quad \mathbb{E}[m(\mathcal{C})] = \frac{p_{\text{in}}}{2} (n(s-1)),$$

and accordingly for coverage and performance

$$\begin{aligned} \mathbb{E}[\text{coverage}(\mathcal{C})] &= \frac{(s-1)p_{\text{in}}}{(s-1)p_{\text{in}} + (n-s)p_{\text{out}}} \\ 1 - \mathbb{E}[\text{performance}(\mathcal{C})] &= \frac{(n-s)p_{\text{out}} + (1-p_{\text{in}})(s-1)}{n-1}. \end{aligned}$$

In the following, we can assume that for our randomly generated instances the initial clustering has the expected behavior with respect to the indices considered.

¹ In case a graph generated that way is not connected, additional edges combining the components are added.

4.2 Technical Details of the Experiments and Implementation

For our experiments, randomly generated instances with the following values of (n, s, v) respectively $p_{\text{in}}, p_{\text{out}}$ are considered. We set $v = 4$ and choose s uniformly at random from $\{\frac{n}{\ell} : 2 \leq \ell \leq \sqrt{n}\}$. Experiments are performed for $n = 100$ and $n = 1000$. On one hand, all combinations of probabilities p_{in} and p_{out} at a distance of 0.05 are considered. On the other hand, for two different values $p_{\text{in}} = 0.4$ and $p_{\text{in}} = 0.75$, p_{out} is chosen such that the ratio of $\overline{m}(\mathcal{C})$ and $m(\mathcal{C})$ for the initial clustering $\mathcal{C}$ is at most 0.5, 0.75 respectively 0.95.

The free parameters of the algorithms are set to $e = 2$ and $r = 2$ in MCL, $\alpha^* = 0.475$ and $\alpha^* = 0.25$ in ICC, and dimension $d = 2$ in GMC. As objective function *quality* in GMC, *coverage*, *performance*, intra-cluster conductance α , inter-cluster conductance δ , as well as the geometric mean of *coverage*, *performance* and δ is considered².

All experiments are repeated at least 30 times and until the maximal length of the confidence intervals is not larger than 0.1 with high probability. The implementation is written in C++ using the GNU compiler g++(2.95.3). We used LEDA 4.3³ and LAPACK++⁴. The experiments were performed on an Intel Xeon with 1.2 ($n = 100$) and 2.4 ($n = 1000$) GHz on the Linux 2.4 platform.

4.3 Computational Results

We concentrate on the behavior of the algorithms with respect to running time, the values for the initial clustering in contrast to the values obtained by the algorithms for the indices under consideration, and the general behavior of the algorithms with respect to the variants of random instances. In addition, we also performed some experiments with grid-like graphs.

Running Time. The experimental study confirms the theoretical statements in Section 3 about the asymptotic worst-case complexity of the algorithms. MCL is significantly slower than ICC and GMC. Not surprisingly as the running time of ICC depends on the number of splittings, ICC is faster for $\alpha^* = 0.25$ than for $\alpha^* = 0.475$. Note that the coarseness of the clustering computed by ICC results from the value of α^* .

For all choices of *quality* except intra-cluster conductance, GMC is the most efficient algorithm. Note that the higher running time of GMC with *quality* set to intra-cluster conductance is only due to the elaborate approximation algorithm for the computation of the intra-cluster conductance value. In summary, GMC with *quality* being the geometric mean of *coverage*, *performance* and inter-cluster conductance, respectively *quality* being an appropriate combination of those indices is the most efficient algorithm under comparison. See Figure 1.

² Experiments considering the geometric mean of all four indices showed that incorporation of intra-cluster conductance did not yield significantly different results. We therefore omit intra-cluster conductance because of efficiency reasons.

³ <http://www.algorithmic-solutions.com>

⁴ <http://www.netlib.org/lapack/>

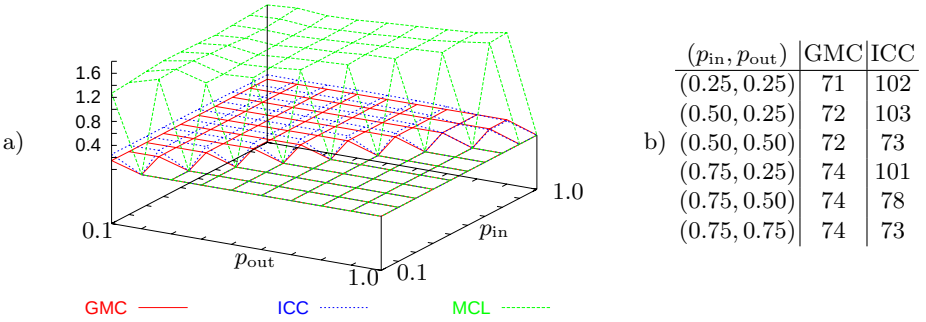


Fig. 1. Running-time in seconds for $n = 100$ (a) and $n = 1000$ (b).

Indices for the Initial Clustering. Studying *coverage*, *performance*, intra- and inter-cluster conductance of the initial clustering gives some useful insights about these indices. Of course, for *coverage* and *performance* the highest values are achieved for the combination of very high p_{in} and very low p_{out} . The *performance* value is greater than the *coverage* value, and the slope of the *performance* level curves remains constant while the slope of the *coverage* level curves decreases with increasing p_{in} . This is because *performance* considers both, edges inside and non-edges between clusters, while *coverage* measures only the fraction of intra-cluster edges within all edges.

The fluctuations of the inter-cluster conductance values for higher values of p_{out} can be explained by the dependency of inter-cluster conductance $\delta(\mathcal{C})$ from the cluster $C_i \in \mathcal{C}$ maximizing ϕ . This shows that inter-cluster conductance is very sensitive to the size of the cut induced by a single small cluster. Due to the procedure how instances are generated for a fixed choice of n , the initial clustering often contains one significantly smaller cluster. For higher values of p_{out} , this cluster has a relatively dense connection to the rest of the graph. So, in many cases it is just this cluster that induces the inter-cluster conductance value.

In contrast to the other three indices, intra-cluster conductance shows a completely different behavior with respect to the choices of p_{in} and p_{out} . Actually, intra-cluster conductance does not depend on p_{out} .

Comparing the Algorithms. A significant observation when comparing the three algorithms with respect to the four indices regards their behavior for dense graphs. All algorithms have a tendency to return a trivial clustering containing only one cluster, even for combinations of p_{in} and p_{out} where p_{in} is significantly higher than p_{out} . This suggests a modification of the algorithms to avoid trivial clusterings. However, for ICC such a modification would be a significant deviation from its intended procedure. The consequences of forcing ICC to split even if

the condition for splitting is violated are not clear at all. On the other hand, the approximation guarantee for intra-cluster conductance is no longer maintained if ICC is prevented from splitting even if the condition for splitting is satisfied. For MCL it is not even clear how to incorporate the restriction to non-trivial clusterings. In contrast, it is easy to modify GMC such that only non-trivial clusterings are computed. Just the maximum and the minimum threshold values τ are ignored.

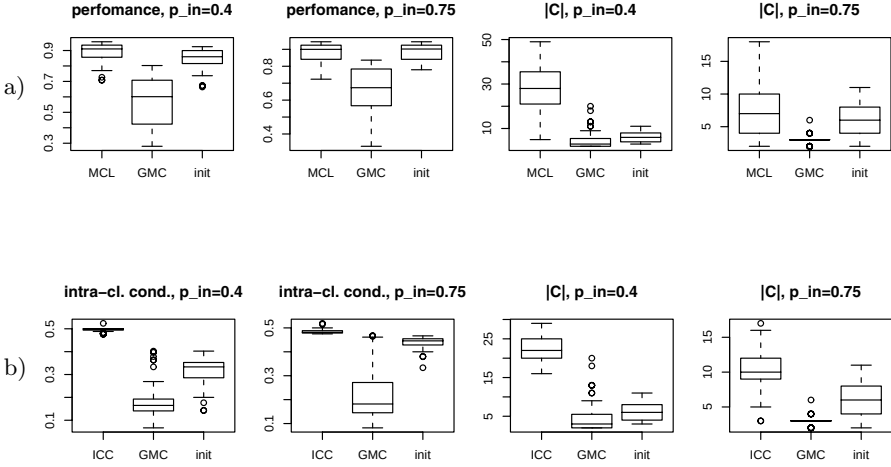


Fig. 2. The diagrams show the distribution of performance respectively intra-cluster conductance and the number of clusters for $p_{in} = 0.4$ respectively $p_{in} = 0.75$, and p_{out} such that at most one third of the edges are inter-cluster edges. The boxes are determined by the first and the third quantile and the internal line represents the median. The shakers extend to 1.5 of the boxes' length (interquartile distance) respectively the extrema. The first two diagrams in 2a) compare the performance values for MCL, GMC and the initial clustering, whereas the last two compare the number of clusters. The first two diagrams in 2b) compare the intra-cluster conductance for MCL, GMC and the initial clustering, whereas the last two compare the number of clusters.

Regarding the cluster indices, MCL does not explicitly target on any of those. However, MCL implicitly targets on identifying loosely connected dense subgraphs. It is argued in [4] that this is formalized by *performance* and that MCL actually yields good results for *performance*. In Figure 2a), the behavior of MCL and GMC are compared with respect to *performance*. The results suggest that MCL indeed performs somewhat better than GMC. The *performance* values for MCL are higher than for GMC and almost identical to the values of the initial clustering. However, MCL has a tendency to produce more clusters than GMC and actually also more than contained in the initial clustering. For instances with high p_{in} , the results for MCL almost coincide with the initial clustering

but the variance is greater. ICC targets explicitly at intra-cluster conductance and its behavior depends on the given α^* . Actually, ICC computes clusterings with intra-cluster conductance α close to α^* . For $\alpha^* = 0.475$, ICC continues the splitting quite long and computes a clustering with many small clusters. In [3] it is argued that *coverage* should be considered together with intra-cluster conductance. However, ICC compares unfavorable with respect to *coverage*. For both choices of α^* , the variation of the *performance* values obtained by ICC is comparable while the resulting values are better for $\alpha^* = 0.475$. This suggests that besides intra-cluster conductance, ICC implicitly targets at *performance* rather than at coverage. Comparing the performance of ICC (with $\alpha^* = 0.475$) and GMC with respect to intra-cluster conductance suggests that ICC is much superior to GMC. Actually, the values obtained by ICC are very similar to the intra-cluster conductance values of the initial clustering. However, studying the number of clusters generated shows that this is achieved at the cost of generating many small clusters. The number of clusters is even significantly bigger than in the initial clustering. This suggests the conclusion that targeting at intra-cluster conductance might lead to unintentional effects. See Figure 2b). Finally, Figure 3 confirms that ICC tends to generate clusterings with many clusters. In contrast, GMC performs very well. It actually generates the ideal clustering.

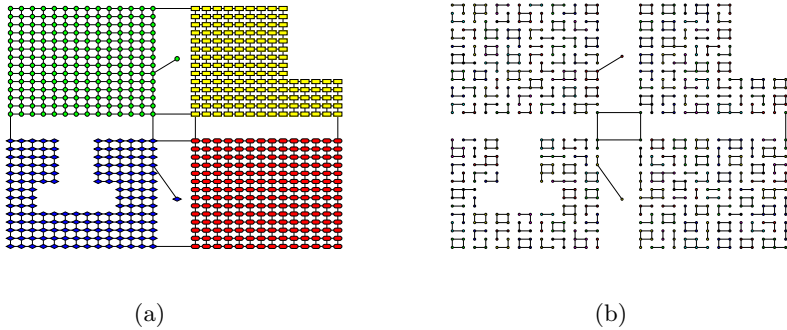


Fig. 3. In 3(a) the clustering determined by GMC for a grid-like graph is shown. The clusters are shown by the different shapes of vertices. In contrast, 3(b) shows the clustering determined by ICC. Inter-cluster edges are not omitted to visualize the clusters.

5 Conclusion

The experimental study confirms the promising expectations about MCL, i.e. in many cases MCL seems to perform well. However, MCL often generates a trivial clustering. Moreover, MCL is very slow. The theoretical result on ICC is reflected by the experimental study, i.e., ICC computes clusterings that are good with respect to intra-cluster conductance. On the other hand, there is the suspect that

the index intra-cluster conductance does not measure the quality of a clustering appropriately. Indeed, the experimental study shows that all four cluster indices have weaknesses. Optimizing only with respect to one of the indices often leads to unintended effects. Considering combinations of those indices is an obvious attempt for further investigations. Moreover, refinement of the embedding used by GMC offers additional potential. So far, only the embedding canonically induced by the eigenvectors is incorporated. By choosing different weightings for the distances in the different dimensions, the effect of the eigenvectors can be controlled. Actually, because of its flexibility with respect to the usage of the geometric clustering and the objective function considered, GMC is superior to MCL and ICC. Finally, because of its small running time GMC is a promising approach for clustering large graphs.

References

1. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice Hall (1988)
2. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. *ACM Computing Surveys* **31** (1999) 264–323
3. Kannan, R., Vampala, S., Vetta, A.: On Clustering — Good, Bad and Spectral. In: *Foundations of Computer Science 2000*. (2000) 367–378
4. van Dongen, S.M.: Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht (2000)
5. Harel, D., Koren, Y.: On clustering using random walks. *Foundations of Software Technology and Theoretical Computer Science* **2245** (2001) 18–41
6. Hartuv, E., Shamir, R.: A clustering algorithm based on graph connectivity. *Information Processing Letters* **76** (2000) 175–181
7. Spielman, D.A., Teng, S.H.: Spectral partitioning works: Planar graphs and finite element meshes. In: *IEEE Symposium on Foundations of Computer Science*. (1996) 96–105
8. Chung, F., Yau, S.T.: Eigenvalues, flows and separators of graphs. In: *Proceeding of the 29th Annual ACM Symposium on Theory of Computing*. (1997) 749
9. Chung, F., Yau, S.T.: A near optimal algorithm for edge separators. In: *Proceeding of the 26th Annual ACM Symposium on Theory of Computing*. (1994) 1–8
10. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., Proietti, M.: Complexity and Approximation – Combinatorial optimization problems and their approximability properties. Springer-Verlag (1999)
11. Wagner, D., Wagner, F.: Between Min Cut and Graph Bisection. In Borzyszkowski, A.M., Sokolowski, S., eds.: *Lecture Notes in Computer Science*, Springer-Verlag (1993) 744–750
12. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. *Theoretical Computer Science* **1** (1976) 237–267
13. Gaertler, M.: Clustering with spectral methods. Master's thesis, Universität Konstanz (2002)
14. Zahn, C.: Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers* **C-20** (1971) 68–86
15. Chung, F.R.K.: Spectral Graph Theory. Number 52 in Conference Board of the Mathematical Sciences. American Mathematical Society (1994)

More Reliable Protein NMR Peak Assignment via Improved 2-Interval Scheduling^{*}

Zhi-Zhong Chen^{1**}, Tao Jiang^{2***}, Guohui Lin^{3†}, Romeo Rizzi⁴,
Jianjun Wen^{2‡}, Dong Xu^{5§}, and Ying Xu^{5§}

¹ Dept. of Math. Sci., Tokyo Denki Univ., Hatoyama, Saitama 350-0394, Japan.
`chen@r.dendai.ac.jp`

² Dept. of Comput. Sci., Univ. of California, Riverside, CA 92521.
`{jiang,wjianju}@cs.ucr.edu`

³ Dept. of Comput. Sci., Univ. of Alberta, Edmonton, Alberta T6G 2E8, Canada.
`ghlin@cs.ualberta.ca`

⁴ Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy.
`romeo@science.unitn.it`

⁵ Life Sciences Division, Oak Ridge National Lab., Oak Ridge, TN 37831.
`{xud,xyn}@ornl.gov`

Abstract. Protein NMR peak assignment refers to the process of assigning a group of “spin systems” obtained experimentally to a protein sequence of amino acids. The automation of this process is still an unsolved and challenging problem in NMR protein structure determination. Recently, protein backbone NMR peak assignment has been formulated as an *interval scheduling* problem, where a protein sequence $\mathcal{P}$ of amino acids is viewed as a discrete time interval $\mathcal{I}$ (the amino acids on $\mathcal{P}$ one-to-one correspond to the time units of $\mathcal{I}$), each subset S of spin systems that are known to originate from consecutive amino acids of $\mathcal{P}$ is viewed as a “job” j_S , the preference of assigning S to a subsequence P of consecutive amino acids on $\mathcal{P}$ is viewed as the profit of executing job j_S in the subinterval of $\mathcal{I}$ corresponding to P , and the goal is to maximize the total profit of executing the jobs (on a single machine) during $\mathcal{I}$. The interval scheduling problem is Max SNP-hard in general. Typically the jobs that require one or two consecutive time units are the most difficult to assign/schedule. To solve these most difficult assignments, we present an efficient $\frac{13}{7}$ -approximation algorithm. Combining this algorithm with a greedy filtering strategy for handling long jobs (*i.e.* jobs that need more than two consecutive time units), we obtained a new efficient heuristic

^{*} The full version can be found at <http://rnc.r.dendai.ac.jp/~chen/papers/pnmr.pdf>

^{**} Supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education of Japan, under Grant No. 14580390.

^{***} Supported in part by NSF Grants CCR-9988353 and ITR-0085910, and National Key Project for Basic Research (973).

[†] Supported in part by NSERC and PENCE, and a Startup Grant from University of Alberta.

[‡] Supported by NSF Grant CCR-9988353.

[§] Supported by the Office of Biological and Environmental Research, U.S. Department of Energy, under Contract DE-AC05-00OR22725, managed by UT-Battelle, LLC.

for protein NMR peak assignment. Our study using experimental data shows that the new heuristic produces the best peak assignment in most of the cases, compared with the NMR peak assignment algorithms in the literature. The $\frac{13}{7}$ -approximation algorithm is also the first approximation algorithm for a nontrivial case of the classical (weighted) interval scheduling problem that breaks the ratio 2 barrier.

1 Introduction

The NMR (nuclear magnetic resonance) technique is a major method to determine protein structures. A time-consuming bottleneck of this technique is the NMR peak assignment, which usually takes weeks or sometimes even months of manual work to produce a nearly complete assignment. A protein NMR peak assignment is to establish a one-to-one mapping between two sets of data: (1) a group of “spin systems” obtained experimentally, each corresponding to a number of spectra related to the same amino acid; (2) a known protein sequence of amino acids. The automation of the assignment process is still an unsolved and challenging problem in NMR protein structure determination.

Two key pieces of information form the foundation of NMR peak assignment and the starting point of our work:

- The likelihood (or weight) of the matching between a spin system and an amino acid on the protein sequence. The weight can be derived from the statistical distribution of spin systems in different amino acid types and predicted secondary structures [12].
- The sequential adjacency (*i.e.*, consecutivity) information of some subsets of spin systems (*i.e.*, each such subset of spin systems should correspond to a subsequence of consecutive amino acids on the protein sequence). Each maximal such subset is called a *segment of spin systems*. It is worth noting that each segment usually consists of at most 10 spin systems. The adjacency information can be obtained from experiments.

In a recently developed computational framework [12], the NMR peak assignment problem has been formulated as a (weighted) *interval scheduling problem*¹ as follows. A protein sequence $\mathcal{P}$ of amino acids is viewed as a discrete time interval $\mathcal{I}$ (the amino acids on $\mathcal{P}$ one-to-one correspond to the time units of $\mathcal{I}$). Each segment S of spin systems is viewed as a job j_S . Each job j_S requires $|S|$ consecutive time units of $\mathcal{I}$ (this corresponds to the requirement that the spin systems in S should be assigned to $|S|$ consecutive amino acids on $\mathcal{P}$). For each time unit t of $\mathcal{I}$, the profit $w(j_S, t)$ of starting job j_S at time unit t and finishing at time unit $t + |S| - 1$ of $\mathcal{I}$ corresponds to the preference (or total weight) of assigning the spin systems in S to those $|S|$ consecutive amino acids on $\mathcal{P}$ that correspond to the time units $t, t + 1, \dots, t + |S| - 1$. Given $\mathcal{I}$, the jobs j_S , and the profits $w(j_S, t)$, our goal is to maximize the total profit of the executed jobs (*i.e.*,

¹ In [12] it was called the *constrained bipartite matching* problem.

we want to find a maximum-likelihood assignment of the given spin systems to the amino acids on $\mathcal{P}$).

Unfortunately, the interval scheduling problem is Max SNP-hard [4,5]. Indeed, for every integer $k \geq 2$, the special case of the interval scheduling problem (called the *k-interval scheduling problem* or *k-ISP* for short), where each job requires at most k consecutive time units, is Max SNP-hard. On the other hand, several 2-approximation algorithms for the interval scheduling problem have been developed [1,2,4,5]. Although these algorithms are theoretically sound, applying them to protein NMR peak assignment produces unsatisfactory assignments as demonstrated in [4]. A major reason why these algorithms do not have good performance in protein NMR peak assignment is that they ignore the following important observation:

- In protein NMR peak assignment, long segments of spin systems are typically easier to assign than shorter ones. Indeed, many long segments have obvious matches based on the total matching weights, while assignments of isolated spin systems or segments consisting of only two spin systems are ambiguous.

The above observation suggests the following heuristic framework for protein NMR peak assignment: first try to assign segments consisting of at least $k + 1$ spin systems for some small integer k (say, $k = 2$), and then solve an instance of *k-ISP*. In [10], we have presented such a heuristic and have shown that it is very effective for protein NMR peak assignment. A major drawback of the heuristic in [10] is that it uses an inefficient branch-and-bound algorithm for *k-ISP*.

In order to improve the efficiency of the heuristic in [10], we present a new approximation algorithm for 2-ISP in this paper. This algorithm achieves an approximation ratio of $\frac{13}{7}$ and is the first approximation algorithm for a nontrivial case of the classical interval scheduling problem that breaks the ratio 2 barrier.² Our algorithm is combinatorial and quite nontrivial – it consists of four separate algorithms and outputs the best solution returned by them. The main tool used in the algorithm design is maximum-weight bipartite matching and careful manipulation of the input instance. Substituting the new algorithm for the branch-and-bound algorithm in the heuristic in [10], we obtain a new heuristic for protein NMR peak assignment.³ We have performed extensive experiments on 70 instances of NMR data derived from 14 proteins to evaluate the performance of our new heuristic in terms of (i) the weight of the assignment and (ii) the number of correctly assigned resonance peaks. The experimental results show that not only does the new heuristic run very fast, it also produces the best peak assignment on most of the instances, compared with the protein NMR peak assignment algorithms in the recent literature [4,5,10,12].

The rest of the paper is organized as follows. The $\frac{13}{7}$ -approximation algorithm for 2-ISP is presented in Section 2. In Section 3, we consider an interesting special

² For *unweighted* ISP where the profit of executing a job at each specific time interval is either 0 or 1 (independent of the job's length), Chuzhoy et al. [6] gave a 1.582-approximation algorithm. In this paper, our interest is in the weighted problem.

³ The program is available to the public upon request to the authors.

profit function in interval scheduling where the profit of executing a job at each specific time interval is either 0 or proportional to the length of the job,⁴ and we present a $(1.5 + \epsilon)$ -approximation algorithm for 2-ISP under this special profit function for any $\epsilon > 0$,⁵ which improves an approximation result in [5]. In Section 4, we describe our new heuristic for protein NMR peak assignment based on the $\frac{13}{7}$ -approximation algorithm for 2-ISP, and give the experimental results. We end this paper with a short discussion in Section 5.

2 A New Approximation Algorithm for 2-ISP

Let $\mathcal{I}$ be the given discrete time interval. Without loss of generality, we may assume that $\mathcal{I} = [0, I]$. Let $\mathcal{J}_1 = \{v_1, v_2, \dots, v_{n_1}\}$ be the given set of jobs requiring one time unit of $\mathcal{I}$. Let $\mathcal{J}_2 = \{v_{n_1+1}, v_{n_1+3}, \dots, v_{n_1+2n_2-1}\}$ be the given set of jobs requiring two contiguous time units of $\mathcal{I}$. Note that $n_1 + n_2$ is the total number of given jobs. For each $1 \leq i \leq I$, let u_i denote the time unit $[i - 1, i]$ of $\mathcal{I}$. Let $U = \{u_i \mid 1 \leq i \leq I\}$. Let $\mathcal{J}'_2 = \{v_{n_1+2}, v_{n_1+4}, \dots, v_{n_1+2n_2}\}$. Let $V = \mathcal{J}_1 \cup \mathcal{J}_2 \cup \mathcal{J}'_2$. We construct an edge-weighted bipartite graph G with color classes U and V as follows: For every $v_j \in \mathcal{J}_1$ and every $u_i \in U$ such that the profit of executing job v_j in time unit u_i is positive, (u_i, v_j) is an edge of G and its weight is the profit. Similarly, for every $v_j \in \mathcal{J}_2$ and every $u_i \in U$ such that the profit of executing job v_j in the two-time units u_i, u_{i+1} is positive, both (u_i, v_j) and (u_{i+1}, v_{j+1}) are edges of G and the total weight of them is the profit.

A *constrained matching* of G is a matching M of G such that for every $u_i \in U$ and every $v_j \in \mathcal{J}_2$, $(u_i, v_j) \in M$ if and only if $(u_{i+1}, v_{j+1}) \in M$. The objective of 2-ISP is equivalent to finding a maximum-weight constrained matching in G . For each edge (u_i, v_j) of G , let $w(u_i, v_j)$ denote the weight of the edge. For convenience, let $w(u_i, v_j) = 0$ for all $(u_i, v_j) \notin E$. For a (constrained or unconstrained) matching M of G , let $w_1(M)$ (respectively, $w_2(M)$) denote the total weight of edges $(u_i, v_j) \in M$ with $v_j \in \mathcal{J}_1$ (respectively, $v_j \in \mathcal{J}_2 \cup \mathcal{J}'_2$); let $w(M) = w_1(M) + w_2(M)$.

Let M^* be a maximum-weight constrained matching in G . In Sections 2.1, 2.3 through 2.5, we will design four algorithms each outputting a constrained matching in G . The algorithm in Section 2.5 is the main algorithm and is quite sophisticated. We will try to find a large constant ϵ such that the heaviest one among the four output matchings is of weight at least $(\frac{1}{2} + \epsilon)w(M^*)$. It will turn out that $\epsilon = \frac{1}{26}$. So, we will fix $\epsilon = \frac{1}{26}$ for the discussions in this section.

2.1 Algorithm 1

This algorithm will output a constrained matching of large weight when $w_2(M^*)$ is relatively large compared with $w_1(M^*)$. We first explain the idea behind the

⁴ This corresponds to a simplified situation in NMR peak assignment, where each spin system has a few equally preferred matching segments of amino acids.

⁵ A simple modification of this algorithm leads to a $(1.5 + \epsilon)$ -approximation algorithm for unweighted 2-ISP.

algorithm. Suppose that we partition the time interval $\mathcal{I}$ into shorter intervals, called *basic intervals*, in such a way that each basic interval, except possibly the first and the last (which may possibly consist of 1 or 2 time units), consists of 3 time units. There are exactly three such partitions of $\mathcal{I}$. Denote them by P_0 , P_1 , and P_2 , respectively. With respect to each P_h with $0 \leq h \leq 2$, consider the problem $\mathcal{Q}_h$ of finding a constrained scheduling which maximizes the total profit of the executed jobs, but subject to the constraint that each basic interval in P_h can be assigned to at most one job and each executed job should be completed within a single basic interval in P_h . It is not so hard to see that each problem $\mathcal{Q}_h$ requires the computation of a maximum-weight (unconstrained) matching in a suitably constructed bipartite graph, and so is solvable in polynomial time.

We claim that among the three problems $\mathcal{Q}_h$, the best one gives a scheduling by which the executed jobs achieve at least a total profit of $\frac{1}{3}w_1(M^*) + \frac{2}{3}w_2(M^*)$. This claim is actually easier to see, if we refer to a more constrained scheduling problem $\mathcal{Q}'_h$ than $\mathcal{Q}_h$ by adding the following constraint:

- For each job $v_j \in \mathcal{J}_1$ and for each basic interval b in P_h , only the *primary* time unit of b can be assigned to v_j , where the *primary* time unit of b , is u_i if b consists of three time units $u_{i-1}u_iu_{i+1}$, is u_1 if b consists of the first two time units u_1u_2 of $\mathcal{I}$, is u_I if b consists of the last two time units $u_{I-1}u_I$ of $\mathcal{I}$, is b itself if b consists of one time unit only.

Consider an optimal (unconstrained) scheduling M^* . For each job $v_j \in \mathcal{J}_2$, if M^* assigns v_j to two time units u_iu_{i+1} , then this assignment of v_j is also valid in exactly two problems among $\mathcal{Q}'_0$, $\mathcal{Q}'_1$, and $\mathcal{Q}'_2$, because there are exactly two indices $h \in \{0, 1, 2\}$ such that some basic interval in P_h contains both time units u_iu_{i+1} . Similarly, for each job $v_j \in \mathcal{J}_1$, if M^* assigns v_j to one time unit u_i , then this assignment of v_j is also valid in at least one problem among $\mathcal{Q}'_0$, $\mathcal{Q}'_1$, and $\mathcal{Q}'_2$, because there is at least one index $h \in \{0, 1, 2\}$ such that u_i is the primary time unit of some basic interval in P_h . Thus, by inheriting from the optimal scheduling M^* , the three problems $\mathcal{Q}'_h$ have more-constrained schedulings M^*_h such that M^*_h is a sub-scheduling of M^* and the three schedulings M^*_h altogether achieve at least a total profit of $w_1(M^*) + 2w_2(M^*)$. Hence, the best more-constrained scheduling among M^*_1 , M^*_2 , and M^*_3 achieves at least a total profit of $\frac{1}{3}w_1(M^*) + \frac{2}{3}w_2(M^*)$. Indeed, we can prove the following better bound which is needed in later sections:

The best more-constrained scheduling among M^*_1 , M^*_2 , and M^*_3 achieves a total profit of at least $\frac{1}{3}w_1(M^*) + \frac{2}{3}w_2(M^*) + \frac{1}{3}(p_1 + p_I)$, where $p_1 = 0$ (respectively, $p_I = 0$) if M^* assigns no job in $\mathcal{J}_1$ to u_1 (respectively, u_I), while p_1 (respectively, p_I) equals the weight of the edge of M^* incident to u_1 (respectively, u_I) otherwise.

To see why we have this better bound, first note that there are exactly two indices $h \in \{0, 1, 2\}$ such that u_1 is the primary time unit of a basic interval in P_h . Similarly, there are exactly two indices $h \in \{0, 1, 2\}$ such that u_I is the primary time unit of a basic interval in P_h . So, the better bound follows.

Lemma 1. *A constrained matching Z_1 in G can be found in $O(I(n_1 + n_2)(I + n_1 + n_2))$ time, whose weight is at least $\frac{1}{3}w_1(M^*) + \frac{2}{3}w_2(M^*) + \frac{1}{3}(p_1 + p_I)$, where $p_1 = 0$ (respectively, $p_I = 0$) if u_1 (respectively, u_I) is not matched to a vertex of $\mathcal{J}_1$ by M^* , while p_1 (respectively, p_I) equals the weight of the edge of M^* incident to u_1 (respectively, u_I) otherwise.*

Corollary 1. *If $w_1(M^*) \leq (\frac{1}{2} - 3\epsilon)w(M^*)$, then $w(Z_1) \geq (\frac{1}{2} + \epsilon)w(M^*)$.*

2.2 Preparing for the Other Three Algorithms

Before running the other three algorithms, we need to compute a maximum-weight unconstrained matching M_{un}^* of G . The unconstrained matching M_{un}^* will be an additional input to the other three algorithms. Therefore, before proceeding to the details of the algorithms, we fix a maximum-weight unconstrained matching M_{un}^* of G . The algorithms in Sections 2.3 through 2.5 will use M_{un}^* in a sophisticated way. First, we use M_{un}^* to define several subsets of U as follows.

- $U_0 = \{u_i \in U \mid u_i \text{ is not matched by } M_{\text{un}}^*\}.$
- $U_1 = \{u_i \in U \mid u_i \text{ is matched to a } v_j \in \mathcal{J}_1 \text{ by } M_{\text{un}}^*\}.$
- $U_{2,1} = \{u_i \in U \mid u_i \text{ is matched to a } v_j \in \mathcal{J}_2 \text{ by } M_{\text{un}}^*\}.$
- $U_{2,2} = \{u_i \in U \mid u_i \text{ is matched to a } v_j \in \mathcal{J}_2' \text{ by } M_{\text{un}}^*\}.$
- $W = \{u_i \in U_1 \mid u_{i-1} \in U_{2,1} \text{ and } u_{i+1} \in U_{2,2}\}.$
- $W_L = \{u_i \in U \mid u_{i+1} \in W\}$ and $W_R = \{u_i \in U \mid u_{i-1} \in W\}.$

In general, if $u_i \in W$, then $u_{i-1} \in W_L$ and $u_{i+1} \in W_R$. Since $W \subseteq U_1$, $W_L \subseteq U_{2,1}$, and $W_R \subseteq U_{2,2}$, no two sets among W , W_L and W_R can intersect.

A common idea behind the forthcoming algorithms is to divide the weights $w_1(M^*)$ and $w_2(M^*)$ into smaller parts, based on the aforementioned subsets of U . Define the smaller parts as follows.

- β_L is the total weight of edges $(u_i, v_j) \in M^*$ with $u_i \in W_L$ and $v_j \in \mathcal{J}_1$.
- β is the total weight of edges $(u_i, v_j) \in M^*$ with $u_i \in W$ and $v_j \in \mathcal{J}_1$.
- β_R is the total weight of edges $(u_i, v_j) \in M^*$ with $u_i \in W_R$ and $v_j \in \mathcal{J}_1$.
- $\bar{\beta} = w_1(M^*) - \beta_L - \beta - \beta_R.$
- α_0 is the total weight of edges $(u_i, v_j) \in M^*$ such that either $v_j \in \mathcal{J}_2$ and $\{u_i, u_{i+1}\} \cap W = \emptyset$, or $v_j \in \mathcal{J}_2'$ and $\{u_{i-1}, u_i\} \cap W = \emptyset$.
- α_1 is the total weight of edges $(u_i, v_j) \in M^*$ such that either $v_j \in \mathcal{J}_2$ and $\{u_i, u_{i+1}\} \cap W \neq \emptyset$, or $v_j \in \mathcal{J}_2'$ and $\{u_{i-1}, u_i\} \cap W \neq \emptyset$.

Lemma 2. $\alpha_0 + \alpha_1 = w_2(M^*)$ and $\beta_L + \beta + \beta_R + \bar{\beta} = w_1(M^*)$.

Now, we are ready to explain how the four algorithms are related. The algorithm in Section 2.3, called Algorithm 2, will output a constrained matching of weight at least $\frac{1}{3}\bar{\beta} + \frac{2}{3}\alpha_0 + \beta + \frac{2}{3}(\beta_L + \beta_R)$. The algorithm in Section 2.4, called Algorithm 3, will output a constrained matching of weight at least $\beta + \bar{\beta} + \alpha_1$. Thus, if $\beta \geq (\frac{1}{6} + \frac{5}{3}\epsilon)w(M^*)$, then Algorithm 2 or 3 will output a constrained matching of weight at least $(\frac{1}{2} + \epsilon)w(M^*)$ (see Corollary 2 below). On the other hand, if $\beta < (\frac{1}{6} + \frac{5}{3}\epsilon)w(M^*)$, then Algorithm 1 or 4 will output a constrained matching of weight at least $(\frac{1}{2} + \epsilon)w(M^*)$ (see Section 2.6).

2.3 Algorithm 2

The idea behind the algorithm is as follows. Removing the vertices in W leaves $|W| + 1$ blocks of U , each of which consists of consecutive vertices of U . For each block b , we use the idea of Algorithm 1 to construct three graphs $G_{b,0}, G_{b,1}, G_{b,2}$. For each $h \in \{0, 1, 2\}$, we consider the graph $\cup_b G_{b,h}$ where b ranges over all blocks, and obtain a new graph G'_h from $\cup_b G_{b,h}$ by adding the vertices of W and the edges $\{u_i, v_j\}$ of G such that $u_i \in W$ and $v_j \in \mathcal{J}_1$. We then compute a maximum-weight (unconstrained) matching in each G'_h , and further convert it to a constrained matching $\bar{M}_h$ of G as in Algorithm 1. The output of Algorithm 2 is the heaviest matching among $\bar{M}_0, \bar{M}_1, \bar{M}_2$. Using Lemma 1, we can prove:

Lemma 3. *A constrained matching Z_2 in G can be found in $O(I(n_1 + n_2)(I + n_1 + n_2))$ time, whose weight is at least $\frac{1}{3}\bar{\beta} + \frac{2}{3}\alpha_0 + \beta + \frac{2}{3}(\beta_L + \beta_R)$.*

2.4 Algorithm 3

We first explain the idea behind Algorithm 3. Suppose that we partition the time interval $\mathcal{I}$ into shorter intervals in such a way that each shorter interval consists of either one time unit or three time units $u_{i-1}u_iu_{i+1}$ where $u_i \in W$. There is only one such partition of $\mathcal{I}$. Further suppose that we want to execute at most one job in each of the shorter intervals, while maximizing the total profit of the executed jobs. This problem can be solved in polynomial time by computing a maximum-weight (unconstrained) matching in a suitably constructed bipartite graph. Similarly to Lemma 1, we can prove that this matching results in a scheduling by which the executed jobs achieve at least a total profit of $\beta + \bar{\beta} + \alpha_1$.

Lemma 4. *A constrained matching Z_3 in G can be found in $O(I(n_1 + n_2)(I + n_1 + n_2))$ time, whose weight is at least $\beta + \bar{\beta} + \alpha_1$.*

Corollary 2. *If $\beta \geq (\frac{1}{6} + \frac{5}{3}\epsilon)w(M^*)$, then $\max\{w(Z_2), w(Z_3)\} \geq (\frac{1}{2} + \epsilon)w(M^*)$.*

2.5 Algorithm 4

The idea behind Algorithm 4 is to convert M_{un}^* to a constrained matching of G . To convert M_{un}^* , we partition $U_1 \cup U_{2,1}$ (respectively, $U_1 \cup U_{2,2}$) into two subsets, none of which contains two vertices u_i and u_{i+1} such that $u_i \in U_{2,1}$ (respectively, $u_{i+1} \in U_{2,2}$). The set of edges of M_{un}^* incident to the vertices of each such subset can be extended to a constrained matching of G . In this way, we obtain four constrained matchings of G . Algorithm 4 outputs the heaviest total weight among the four matchings. We can prove that the weight of the output matching is at least $w(M_{\text{un}}^*)/2$.

We next proceed to the details of Algorithm 4. Algorithm 4 computes a constrained matching in G as follows.

1. Starting at u_1 , divide U into segments each of which is in the following form:

$$u_{i-\ell}u_{i-\ell+1} \cdots u_{i-1}u_iu_{i+1} \cdots u_{i+r-1}u_{i+r},$$

where $u_j \in U_{2,1}$ for all $i - \ell \leq j \leq i - 1$, $u_j \in U_{2,2}$ for all $i + 1 \leq j \leq i + r$, $u_{i-\ell-1} \notin U_{2,1}$, $u_{i+r+1} \notin U_{2,2}$, and u_i has no restriction. Note that ℓ and/or r may be equal to zero. We call u_i the *center* of the segment. For each segment s , let $c(s)$ denote the integer i such that u_i is the center of s ; let $\ell(s)$ denote the number of vertices in s that precede $u_{c(s)}$; let $r(s)$ denote the number of vertices in s that succeed $u_{c(s)}$.

2. For each segment s , compute two integers x_s and y_s as follows:

- If $u_{c(s)} \in U_0$, then $x_s = c(s) - 1$ and $y_s = c(s) + 1$.
- If $u_{c(s)} \in U_1$, then $x_s = y_s = c(s)$.
- If $u_{c(s)} \in U_{2,1}$, then $x_s = c(s)$ and $y_s = c(s) + 1$.
- If $u_{c(s)} \in U_{2,2}$, then $x_s = c(s) - 1$ and $y_s = c(s)$.

3. Let $U_{2,1}^e = \bigcup_s \{u_i \mid (x_s - i) \bmod 2 = 0, c(s) - \ell(s) \leq i \leq x_s\}$,
 $U_{2,1}^o = \bigcup_s \{u_i \mid (x_s - i) \bmod 2 = 1, c(s) - \ell(s) \leq i \leq x_s\}$,
 $U_{2,2}^e = \bigcup_s \{u_i \mid (i - y_s) \bmod 2 = 0, y_s \leq i \leq c(s) + r(s)\}$,
 $U_{2,2}^o = \bigcup_s \{u_i \mid (i - y_s) \bmod 2 = 1, y_s \leq i \leq c(s) + r(s)\}$,

where s runs over all segments.

4. Let $M_{2,1}^e = \{(u_i, v_j) \in M_{\text{un}}^* \mid u_i \in U_{2,1}^e\} \cup \{(u_{i+1}, v_{j+1}) \mid u_i \in U_{2,1}^e \cap U_{2,1} \text{ and } \{u_i, v_j\} \in M_{\text{un}}^*\}$, $M_{2,1}^o = \{(u_i, v_j) \in M_{\text{un}}^* \mid u_i \in U_{2,1}^o\} \cup \{(u_{i+1}, v_{j+1}) \mid u_i \in U_{2,1}^o \cap U_{2,1} \text{ and } \{u_i, v_j\} \in M_{\text{un}}^*\}$, $M_{2,2}^e = \{(u_i, v_j) \in M_{\text{un}}^* \mid u_i \in U_{2,2}^e\} \cup \{(u_{i-1}, v_{j-1}) \mid u_i \in U_{2,2}^e \cap U_{2,2} \text{ and } \{u_i, v_j\} \in M_{\text{un}}^*\}$, $M_{2,2}^o = \{(u_i, v_j) \in M_{\text{un}}^* \mid u_i \in U_{2,2}^o\} \cup \{(u_{i-1}, v_{j-1}) \mid u_i \in U_{2,2}^o \cap U_{2,2} \text{ and } \{u_i, v_j\} \in M_{\text{un}}^*\}$.
5. For the set $\bar{U}_{2,1}^o$ of vertices of U that are not matched by $M_{2,1}^o$, compute a maximum-weight matching $N_{2,1}^o$ between vertices in $\bar{U}_{2,1}^o$ and vertices in $\mathcal{J}_1$.
6. For the set $\bar{U}_{2,2}^o$ of vertices of U that are not matched by $M_{2,2}^o$, compute a maximum-weight matching $N_{2,2}^o$ between vertices in $\bar{U}_{2,2}^o$ and vertices in $\mathcal{J}_1$.
7. Output the maximum-weight matching Z_4 among $M_{2,1}^e$, $M_{2,1}^o \cup N_{2,1}^o$, $M_{2,2}^e$, $M_{2,2}^o \cup N_{2,2}^o$.

Lemma 5. $M_{2,1}^e$, $M_{2,1}^o \cup N_{2,1}^o$, $M_{2,2}^e$ and $M_{2,2}^o \cup N_{2,2}^o$ are constrained matchings.

Lemma 6. $w(M_{2,1}^e) + w(M_{2,1}^o) + w(M_{2,2}^e) + w(M_{2,2}^o) \geq 2w(M_{\text{un}}^*)$.

Lemma 7. $(U - \bar{U}_{2,1}^o) \cap (U - \bar{U}_{2,2}^o) \subseteq W$.

2.6 Performance of the Algorithm When β Is Small

For a contradiction, assume the following:

Assumption 1 $\beta < (\frac{1}{6} + \frac{5}{3}\epsilon)w(M^*)$ and $\max\{w(Z_1), w(Z_4)\} < (\frac{1}{2} + \epsilon)w(M^*)$.

We want to derive a contradiction under this assumption. First, we derive three inequalities from this assumption and the lemmas in Section 2.5.

Lemma 8. $w(M_{2,1}^o) + w(M_{2,2}^o) \geq (1 - 2\epsilon)w(M^*)$.

Lemma 9. $w(N_{2,1}^o) + w(N_{2,2}^o) < 4\epsilon w(M^*)$.

Lemma 10. $\beta > w_1(M^*) - 4\epsilon w(M^*)$.

Now, we are ready to get a contradiction. By Corollary 1 and Assumption 1, $w_1(M^*) > (\frac{1}{2} - 3\epsilon)w(M^*)$. Thus, by Lemma 10, $\beta > (\frac{1}{2} - 7\epsilon)w(M^*)$. On the other hand, by Assumption 1, $\beta < (\frac{1}{6} + \frac{5}{3}\epsilon)w(M^*)$. Hence, $\frac{1}{2} - 7\epsilon < \frac{1}{6} + \frac{5}{3}\epsilon$, contradicting our choice that $\epsilon = \frac{1}{26}$. Therefore,

Theorem 1. *A constrained matching Z in G with $w(Z) \geq \frac{13}{7}w(M^*)$ can be found in $O(I(n_1 + n_2)(I + n_1 + n_2))$ time.*

3 2-ISP with a Special Profit Function

In this section, we consider *proportional 2-ISP*, where the profit of executing a job at each specific time interval is either 0 or proportional to the length of the job. A $\frac{5}{3}$ -approximation algorithm was recently presented in [5] for proportional 2-ISP. Here, we present a $(1.5 + \epsilon)$ -approximation algorithm for it for any $\epsilon > 0$. We note in passing that a simple modification of this algorithm leads to a $(1.5 + \epsilon)$ -approximation algorithm for unweighted 2-ISP.

Let U , $\mathcal{J}_1$, and $\mathcal{J}_2$ be as in Section 2. Let E be the set of those $(u_i, v_j) \in U \times \mathcal{J}_1$ such that the profit of executing job v_j in time unit u_i is positive. Let F be the set of those $(u_i, u_{i+1}, v_j) \in U \times U \times \mathcal{J}_2$ such that the profit of executing job v_j in time units u_i and u_{i+1} is positive.

Consider the hypergraph $H = (U \cup \mathcal{J}_1 \cup \mathcal{J}_2, E \cup F)$ on vertex set $U \cup \mathcal{J}_1 \cup \mathcal{J}_2$ and on edge set $E \cup F$. Obviously, proportional 2-ISP becomes the problem of finding a matching $E' \cup F'$ in H with $E' \subseteq E$ and $F' \subseteq F$ such that $|E'| + 2|F'|$ is maximized over all matchings in H . Our idea is to reduce this problem to the problem of finding a maximum cardinality matching in a 3-uniform hypergraph (*i.e.* each hyperedge consists of exactly three vertices). Since the latter problem admits a $(1.5 + \epsilon)$ -approximation algorithm [7] and our reduction is approximation preserving, it follows that proportional 2-ISP admits a $(1.5 + \epsilon)$ -approximation algorithm.

Theorem 2. *For every $\epsilon > 0$, there is a polynomial-time $(1.5 + \epsilon)$ -approximation algorithm for proportional 2-ISP.*

4 A New Heuristic for Protein NMR Peak Assignment

As mentioned in Section 1, the $\frac{13}{7}$ -approximation algorithm for 2-ISP can be easily incorporated into a heuristic framework for protein NMR peak assignment introduced in [10]. The heuristic first tries to assign “long” segments of three or more spin systems that are under the consecutivity constraint to segments of the

host protein sequence, using a simple greedy strategy, and then solves an instance of 2-ISP formed by the remaining unassigned spin systems and amino acids. The first step of the framework is also called *greedy filtering* and may potentially help improve the accuracy of the heuristic significantly in practice because we are often able to assign long segments of spin systems with high confidence. We have tested the new heuristic based on the $\frac{13}{7}$ -approximation algorithm for 2-ISP and compared the results with two of the best approximation and heuristic algorithms in [4,5,10], namely the 2-approximation algorithm for the interval scheduling problem [4,5] and the branch-and-bound algorithm (augmented with greedy filtering) [10]⁶. The test data consists of 70 (pseudo) real instances of NMR peak assignment derived from 14 proteins. For each protein, the data of spin systems were from the experimental data in the BioMagResBank database [11], while 5 (density) levels of consecutivity constraints were simulated, as shown in Table 1.

Note that, both the new heuristic algorithm and the 2-approximation algorithm are very fast in general while the branch-and-bound algorithm can be much slower because it may have to explore much of the entire search space. On a standard Linux workstation, it took seconds to hours for each assignment by the branch-and-bound algorithm in the above experiment, while it took a few seconds consistently using either the new heuristic algorithm or the 2-approximation algorithm. Table 1 shows the comparison of the performance of the three algorithms in terms of (i) the weight of the assignment and (ii) the number of correctly assigned spin systems. Although measure (i) is the objective in the interval scheduling problem, measure (ii) is what it counts in NMR peak assignment. Clearly, the new heuristic outperformed the 2-approximation algorithm in both measures by large margins. Furthermore, the new heuristic outperformed the branch-and-bound algorithm in measure (ii), although the branch-and-bound algorithm did slightly better in measure (i). More precisely, the new heuristic was able to assign the same number of or more spin systems correctly than the branch-and-bound algorithm on 53 out of the 70 instances, among which the new heuristic algorithm improved over the branch-and-bound algorithm on 39 instances.⁷ Previously, the branch-and-bound algorithm was known to have the best assignment accuracy (among all heuristics proposed for the interval scheduling problem) [10]. The result demonstrates that this new heuristic based on the $\frac{13}{7}$ -approximation algorithm for 2-ISP will be very useful in the automation of NMR peak assignment. In particular, the good assignment accuracy and fast speed allow us to tackle some large-scale problems in experimental NMR peak assignment within realistic computation resources. As an example of application, the consecutivity information derived from experiments may sometimes be ambiguous. The new heuristic algorithm makes it possible for the user to experiment with different interpretations of consecutivity and compare the resulting assignments.

⁶ It is worth mentioning that other automated NMR peak assignment programs generally require more NMR experiments, and so they cannot be compared with ours.

⁷ It is not completely clear to us why the new heuristic did better on these 39 instances.

Table 1. The performance of the new heuristic comprising greedy filtering and the $\frac{13}{7}$ -approximation algorithm for 2-ISP in comparison with two of the best approximation and heuristic algorithms in [4,5,10] on 70 instances of NMR peak assignment. The protein is represented by the entry name in the BioMagResBank database [11], e.g., bmr4752. The number after the underscore symbol indicates the density level of consecutivity constraints, e.g., $_{.6}$ means that 60% of the spin systems are connected to form segments. W_1 and R_1 represent the total assignment weight and number of spin systems correctly assigned by the new heuristic, respectively. W_2 and R_2 (W_3 and R_3) are corresponding values for the 2-approximation algorithm for the interval scheduling problem (the branch-and-bound algorithm augmented with greedy filtering, respectively). The numbers in bold indicate that all the spin systems are correctly assigned. The total numbers of spin systems in other proteins are 158 for bmr4027, 215 for bmr4318, 78 for bmr4144, 115 for bmr4302, and 156 for bmr4393.

	W_1	R_1	W_2	R_2	W_3	R_3		W_1	R_1	W_2	R_2	W_3	R_3
bmr4027.5	1873820	40	1827498	3	1934329	33	bmr4144.5	919419	11	921816	17	997603	16
bmr4027.6	1854762	64	1818131	8	1921093	37	bmr4144.6	923546	21	897500	11	993361	11
bmr4027.7	1845477	89	1784027	44	1910897	74	bmr4144.7	954141	68	842073	2	954633	64
bmr4027.8	1900416	151	1671475	19	1894532	128	bmr4144.8	953741	69	804531	5	954585	67
bmr4027.9	1896606	156	1652859	60	1896606	156	bmr4144.9	952241	75	837519	35	952241	75
bmr4288.5	1243144	36	1169907	6	1255475	12	bmr4302.5	1275787	31	1219920	11	1331391	16
bmr4288.6	1197106	49	1179110	15	1261696	26	bmr4302.6	1282789	51	1174564	0	1324395	43
bmr4288.7	1232771	65	1112288	22	1251020	57	bmr4302.7	1310324	78	1181267	8	1323495	62
bmr4288.8	1201192	68	1133554	35	1238344	66	bmr4302.8	1308217	112	1152323	27	1308217	103
bmr4288.9	1249465	105	1051817	48	1249465	105	bmr4302.9	1250300	111	1293954	107	1298321	110
bmr4309.5	1974762	35	1954955	13	2117910	25	bmr4316.5	999920	43	890944	2	1009329	30
bmr4309.6	1960424	48	1924727	12	2110992	57	bmr4316.6	967526	59	863207	13	1022505	35
bmr4309.7	2046029	119	1885986	24	2093595	77	bmr4316.7	925817	75	882818	9	1029287	79
bmr4309.8	1962114	121	1868338	55	2067295	101	bmr4316.8	1005898	75	957378	62	1029287	89
bmr4309.9	2048987	178	1796864	95	2048987	178	bmr4316.9	1029827	89	984774	85	1029287	89
bmr4318.5	2338383	19	2355926	2	2497294	20	bmr4353.5	1468772	20	1417351	8	1532518	17
bmr4318.6	2265090	34	2312260	13	2481789	35	bmr4353.6	1428944	23	1421633	18	1524784	24
bmr4318.7	2268700	73	2259377	52	2444439	52	bmr4353.7	1461648	56	1370235	14	1516244	44
bmr4318.8	2217936	92	2214174	63	2420829	62	bmr4353.8	1443261	78	1337329	9	1472871	80
bmr4318.9	2339582	201	2158223	122	2383453	201	bmr4353.9	1474022	124	1273988	15	1483781	126
bmr4391.5	691804	10	688400	5	753046	18	bmr4393.5	1816837	49	1742954	3	1874095	41
bmr4391.6	680959	7	699066	8	745501	10	bmr4393.6	1843685	71	1772955	42	1871616	59
bmr4391.7	699199	17	684953	37	735683	26	bmr4393.7	1847874	102	1722026	22	1862221	76
bmr4391.8	688368	38	663147	30	723111	42	bmr4393.8	1832576	129	1709538	65	1853749	130
bmr4391.9	710914	66	687290	45	710914	66	bmr4393.9	1837340	142	1527885	3	1851298	152
bmr4579.5	913713	18	894084	2	967647	15	bmr4670.5	1365873	32	1309727	11	1435721	22
bmr4579.6	889118	35	911564	8	976720	32	bmr4670.6	1326082	35	1290812	13	1429449	30
bmr4579.7	903586	48	873884	17	958335	44	bmr4670.7	1353618	78	1239001	6	1402335	38
bmr4579.8	933371	72	877556	26	956115	63	bmr4670.8	1391055	116	1236726	19	1391055	116
bmr4579.9	950173	86	760356	0	950173	86	bmr4670.9	1391055	120	1237614	60	1391055	116
bmr4752.5	881020	21	796019	8	884307	21	bmr4929.5	1410017	17	1408112	4	1496460	23
bmr4752.6	877313	32	824289	6	892520	32	bmr4929.6	1391418	36	1385673	12	1496954	32
bmr4752.7	866896	43	752633	3	887292	41	bmr4929.7	1427122	69	1378166	30	1490155	56
bmr4752.8	882755	68	730276	17	882755	68	bmr4929.8	1459368	82	1281548	18	1481593	88
bmr4752.9	882755	68	812950	44	882755	68	bmr4929.9	1477704	114	1178499	20	1477704	114

5 Discussion

The computational method, presented in this paper, provides a more accurate and more efficient technique for NMR peak assignment, compared to our previous algorithms [4,5,10,12]. We are in the process of incorporating this algorithm

into a computational pipeline for fast protein fold recognition and structure determination, using an iterative procedure of NMR peak assignments and protein structure prediction. The basic idea of this pipeline is briefly outlined as follows.

Recent developments in applications of *residual dipolar coupling* (RDC) data to protein structure determination have indicated that RDC data alone may be adequate for accurate resolution of protein structures [8], bypassing the expensive and time-consuming step of NOE (nuclear Overhauser effect) data collection and assignments. We have recently demonstrated (unpublished results) that if the RDC data/peaks are accurately assigned, we can accurately identify the correct fold of a target protein in the PDB database [3] even when the target protein has lower than 25% of sequence identity with the corresponding PDB protein of the same structural fold. In addition, we have found that RDC data can be used to accurately rank sequence-fold alignments (alignment accuracy), suggesting the possibility of protein backbone structure prediction by combining RDC data and fold-recognition techniques like protein threading [13].

By including RDC data in our peak assignment algorithm (like [9]), we expect to achieve two things: (a) an improved accuracy of peak assignments with the added information, and (b) an assignment (possibly partial) of the RDC peaks. Using assigned RDC peaks and the aforementioned strategy, we can identify the correct structural folds of a target protein in the PDB database. Then based on the identified structural fold and a computed sequence-fold alignment, we can back-calculate the theoretical RDC peaks of the predicted backbone structure. Through matching the theoretical and experimental RDC peaks, we can establish an iterative procedure for NMR data assignment and structure prediction. Such a process will iterate until most of the RDC peaks are assigned and a structure is predicted. We expect that such a procedure will prove to be highly effective for fast and accurate protein fold and backbone structure predictions, using NMR data from only a small number of NMR experiments.

References

1. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48:1069–1090, 2001.
2. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *Proceedings of STOC'99*, 622–631.
3. F. C. Bernstein, T. F. Koetzle, G. J. B. Williams, E. F. Meyer, M. D. Brice, J. R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi, The Protein Data Bank: A Computer Based Archival File for Macromolecular Structures, *J. Mol. Biol.*, 112:535–542, 1977.
4. Z.-Z. Chen, T. Jiang, G. Lin, J. Wen, D. Xu, J. Xu, and Y. Xu. Approximation algorithms for NMR spectral peak assignment. *TCS*, 299:211–229, 2003.
5. Z.-Z. Chen, T. Jiang, G. Lin, J. Wen, D. Xu, and Y. Xu. Improved approximation algorithms for NMR spectral peak assignment. *Proceedings of WABI'2002*, 82–96.
6. J. Chuzhoy, R. Ostrovsky, and Y. Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *FOCS'2001*, 348–356.

7. C.A.J. Hurkens and A. Schrijver. On the size of systems of sets of every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.
8. J.C. Hus, D. Marion and M. Blackledge, Determination of protein backbone structure using only residual dipolar couplings, *J. Am. Chem. Soc.*, 123:1541–1542, 2001.
9. J.C. Hus, J.J. Prompers, and R. Bruschweiler, Assignment strategy for proteins with known structure, *Journal of Magnetic Resonance*, 157:119–123, 2002.
10. G. Lin, D. Xu, Z.-Z. Chen, T. Jiang, J. Wen, and Y. Xu. Computational assignments of protein backbone NMR peaks by efficient bounding and filtering. *Journal of Bioinformatics and Computational Biology*, 31:944–952, 2003.
11. University of Wisconsin. *BioMagResBank*. <http://www.bmrb.wisc.edu>. University of Wisconsin, Madison, Wisconsin, 2001.
12. Y. Xu, D. Xu, D. Kim, V. Olman, J. Razumovskaya, and T. Jiang. Automated assignment of backbone NMR peaks using constrained bipartite matching. *IEEE Computing in Science & Engineering*, 4:50–62, 2002.
13. Y. Xu and D. Xu, Protein Threading using PROSPECT: design and evaluation, *Protein: Structure, Function, Genetics*, 40:343–354, 2000.

The Minimum Shift Design Problem: Theory and Practice

Luca Di Gaspero¹, Johannes Gärtner², Guy Kortsarz³, Nysret Musliu⁴,
Andrea Schaerf⁵, and Wolfgang Slany⁶

¹ University of Udine, Italy, digasper@dimi.uniud.it

² Ximes Inc, Austria, gaertner@ximes.com

³ Rutgers University, USA, guyk@camden.rutgers.edu

⁴ Technische Universität Wien, Austria, musliu@dbai.tuwien.ac.at

⁵ University of Udine, Italy, schaerf@uniud.it

⁶ Technische Universität Graz, Austria, wsi@ist.tugraz.at

Abstract. We study the minimum shift design problem (*MSD*) that arose in a commercial shift scheduling software project: Given a collection of shifts and workforce requirements for a certain time interval, we look for a minimum cardinality subset of the shifts together with an optimal assignment of workers to this subset of shifts such that the deviation from the requirements is minimum. This problem is closely related to the minimum edge-cost flow problem (*MECF*), a network flow variant that has many applications beyond shift scheduling. We show that *MSD* reduces to a special case of *MECF*. We give a logarithmic hardness of approximation lower bound. In the second part of the paper, we present practical heuristics for *MSD*. First, we describe a local search procedure based on interleaving different neighborhood definitions. Second, we describe a new greedy heuristic that uses a min-cost max-flow (*MCMF*) subroutine, inspired by the relation between the *MSD* and *MECF* problems. The third heuristic consists of a serial combination of the other two. An experimental analysis shows that our new heuristics clearly outperform an existing commercial implementation.

1 Introduction

The minimum shift design problem (*MSD*) concerns selecting which work shifts to use, and how many people to assign to each shift, in order to meet prespecified staffing requirements.

The *MSD* problem arose in a project at Ximes Inc, a consulting and software development company specializing in shift scheduling. The goal of this project was, among others, producing a software end-product called OPA (short for ‘OPerating hours Assistant’). OPA was introduced mid 2001 to the market and has since been successfully sold to end-users besides of being heavily used in the day to day consulting work of Ximes Inc at customer sites (mainly European, but Ximes recently also won a contract with the US ministry of transportation). OPA has been optimized for “presentation”-style use where solutions to many variants

of problem instances are expected to be more or less immediately available for graphical exploration by the audience. Speed is of crucial importance to allow for immediate discussion in working groups and refinement of requirements. Without quick answers, understanding of requirements and consensus building would be much more difficult. OPA and the underlying heuristics have been described in [13,24].

The staffing requirements are given for h days, which usually span a small multiple of a week, and are valid for a certain amount of time ranging from a week up to a year, typically consisting of several months (in the present paper, we disregard the problem of connecting several such periods, though this is handled in OPA). Each day j is split into n equal-size smaller intervals, called *timeslots*, which can last from a few minutes up to several hours. The staffing requirement for the i th timeslot ($i = 0, \dots, n - 1$) on day $j \in \{0, \dots, h - 1\}$ starting at t_i , namely $[t_i, t_{i+1})$, is fixed. For every i and j we are given an integer value $b_{i,j}$ representing the number of persons needed at work from time t_i until time t_{i+1} on day j , with cyclic repetitions after h days. Table 1 shows an example of workforce requirements with $h = 7$, in which, for conciseness, timeslots with same requirements are grouped together (adapted from a real call-center).

Table 1. Sample workforce requirements.

Start	End	Mon	Tue	Wen	Thu	Fri	Sat	Sun
06:00	08:00	2	2	2	6	2	0	0
08:00	09:00	5	5	5	9	5	3	3
09:00	10:00	7	7	7	13	7	5	5
10:00	11:00	9	9	9	15	9	7	7
11:00	14:00	7	7	7	13	7	5	5
14:00	16:00	10	9	7	9	10	5	5
16:00	17:00	7	6	4	6	7	2	2
17:00	22:00	5	4	2	2	5	0	0
22:00	06:00	5	5	5	5	5	5	5

When designing shifts, not all starting times are feasible, neither is any length allowed. The input thus also includes a collection of *shift types*. A shift type has minimum and maximum start times, and minimum and maximum length. Table 2 shows a typical example of the set of shift types. Each shift $I_{s,l}$ with starting time t_s with $s \in \{0, \dots, n - 1\}$ and length l , belongs to a type, i.e., its length and starting times must necessarily be inside the intervals defined by one type. The shift types determine the m available shifts. Assuming a timeslot of length 15 minutes, there are $m = 324$ different shifts belonging to the types of Table 2. The type of shift I is denoted by $T(I)$.

The goal is to decide how many persons $x_j(I_{s,l})$ are going to work in each shift $I_{s,l}$ each day j so that $b_{i,j}$ people will be present at time $[t_i, t_{i+1})$ for all i

Table 2. Typical set of shift types.

Shift type	Possible start times	Possible length
M (morning)	06:00 – 08:00	7h – 9h
D (day)	09:00 – 11:00	7h – 9h
A (afternoon)	13:00 – 15:00	7h – 9h
N (night)	22:00 – 24:00	7h – 9h

and j . Many of the shifts are never used, hence for an unused shift I , $x_j(I) = 0$ for all j .

Let $\mathcal{I}_{t_i}$ be the collection of shifts that include t_i . A feasible solution gives h numbers $x_j(I)$ to each shift $I = I_{s,l}$ so that $p_{i,j} \stackrel{\text{def}}{=} \sum_{I \in \mathcal{I}_{t_i}} x_j(I) = b_{i,j}$, namely, the number of workers present at time t_i for all values of $i \in \{0, \dots, n-1\}$ for all days $j \in \{0, \dots, h-1\}$ meets the staffing requirements. This constraint is usually relaxed such that small deviations are allowed.

Note that a better fit of the requirements might sometimes be achieved by looking for solutions covering more than one cycle of h days. Since this could easily be handled by extending the proposed heuristics or, even simpler, by repeating the requirements for a corresponding number of times, additionally is only very seldomly considered in practice, and theoretically adds nothing to the problem, we do not consider it in this paper.

We now discuss the quality of solutions, i.e. the *objective function* to minimize. When we allow small deviations to the requirements, there are three main objective components. The first and second are, naturally, the staffing excess and shortage, namely, the sums $ex \stackrel{\text{def}}{=} \sum_{i,j} (\max(0, p_{i,j} - b_{i,j}))$ and $sh \stackrel{\text{def}}{=} \sum_{i,j} (\max(0, b_{i,j} - p_{i,j}))$. The third component is the number of shifts selected. Once a shift is selected (at least one person works in this shift during any day) it is not really important how many people work at this shift nor on how many days the shift is reused. However, it is important to have only few shifts as they lead to schedules that have a number of advantages, e.g., if one tries to keep teams of persons together. Such teambuilding may be necessary due to managerial or qualification reasons. While teams are of importance in many but not all schedules, there are further advantages of fewer shifts. With fewer shifts, schedules are easier to design (with or without software support, see [23]). Fewer shifts also make such schedules easier to read, check, manage and administer; each of these activities being a burden in itself. In practice, a number of further optimization criteria clutters the problem, e.g., the average number of working days per week = duties per week. This number is an extremely good indicator with respect to how difficult it will be to develop a schedule and what quality that schedule will have. The average number of duties thereby becomes the key criterion for working conditions and is sometimes even part of collective agreements, e.g., setting 4.81 as the maximum. Fortunately, this and most further criteria can easily be handled by straightforward extensions of the

heuristics described in this paper and add nothing to the complexity of *MSD*. We therefore concentrate on the three main criteria mentioned at the beginning of this paragraph.

In summary, we look for an assignment $x_j(I)$ to all the possible shifts that minimizes an objective function composed by a weighted sum of ex , sh and the number of used shifts, in which the weights depend on the instance.

Table 3. A solution for the problem of Table 1.

Start	Length	Mon	Tue	Wen	Thu	Fri	Sat	Sun
06:00	8h	2	2	2	6	2		
08:00	8h	3	3	3	3	3	3	3
09:00	8h	2	2	2	4	2	2	2
14:00	8h	5	4	2	2	5		
22:00	8h	5	5	5	5	5	5	5

A typical solution for the problem from Table 1 that uses 5 shifts is given in Table 3. Note that there is a shortage of 2 workers every day from 10h–11h that cannot be compensated without having more shortage or excess. Also note that using less than 5 shifts leads to more shortage or excess.

In Section 2 we show a relation of *MSD* to the minimum edge-cost flow (*MECF*) problem (listed as [ND32] in [10]). In this problem the edges in the flow network have a capacity $c(e)$ and a fixed usage cost $p(e)$. The goal is to find a maximum flow function f (obeying the capacity and flow conservation laws) so that the cost $\sum_{e: f(e)>0} p(e)$ of edges carrying non-zero flow is minimized.

This problem is one of the more fundamental flow variants with many applications. A sample of these applications include optimization of synchronous networks (see [21]), source-location (see [3]), transportation (see [8,14,22]), scheduling (for example, trucks or manpower, see [8,20]), routing (see [16]), and designing networks (for example, communication networks with fixed cost per link used, e.g., leased communication lines, see [16,17]).

The *UDIF* (infinite capacities flow on a DAG) problem restricts the *MECF* problem as follows:

1. Every edge not touching the sink or the source has infinite capacity. We call an edge *proper* if it does not touch the source or the sink. Non-proper edges, namely edges touching the source or the sink, have no restriction. Namely, they have arbitrary capacities.
2. The costs of proper edges is 1. The cost of edges touching the source or sink is zero.
3. The underlying flow network is a DAG (directed acyclic graph).
4. The goal is, as in the general problem, to find a maximum flow $f(e)$ over the edges (obeying the capacity and flow conservation laws) and among all

maximum flows to choose the one minimizing the cost of edges carrying non-zero flow. Hence, in this case, minimize the *number* of proper edges carrying nonzero flow (namely, minimizing $|\{e : f(e) > 0, e \text{ is proper}\}|$).

Related Work

Flow-related work: It is well known that finding a maximum flow minimizing $\sum_e p(e)f(e)$ is a polynomial problem, namely, the well known min-cost max-flow problem (see, e.g., [25]). Krumke *et al* [18] studied the approximability of *MECF*. They show that, unless $NP \subseteq DTIME(n^{O(\log \log n)})$, for any $\epsilon > 0$ there can be no approximation algorithm on bipartite graphs with a performance guarantee of $(1 - \epsilon) \ln F$, and also provide an F -ratio approximation algorithm for the problem on general graphs, where F is the flow value. [7] point out a $\beta(G) + 1 + \epsilon$ approximation algorithm for the same problem where $\beta(G)$ is the cardinality of the maximum size bond of G , a bond being a minimal cardinality set of edges whose removal disconnects a pair of vertices with positive demand. A large body of work is devoted to hard variants of the maximum flow problem. For example, the non-approximability of flows with priorities was studied in [6]. In [12] a 2-ratio approximation is given for the NP-hard problem of multicommodity flow in trees. The same authors [11] study the related problem of multicuts in general graphs. In [9] the hardness result for the Minimum Edge Cost Flow Problem (*MECF*) is improved. This paper proves that *MECF* does not admit a $2^{\log^{1-\epsilon} n}$ -ratio approximation, for every constant $\epsilon > 0$, unless $NP \subseteq DTIME(n^{\text{polylog} n})$. The same paper also presents a bi-criteria approximation algorithm for *UDIF*, essentially giving an n^ϵ approximation for the problem for every ϵ .

Work on shift scheduling: There is a large body on shift scheduling problems (see [19] for a recent survey). The larger body of the work is devoted to the case where the shifts are already chosen and what is needed is to allocate the resources to shifts, for which network flow techniques have, among others, been applied. [5] note that a problem similar to *MSD* where the requirement to minimize the number of selected shifts is dropped and there are linear costs for understaffing and overstaffing can be transformed into a min-cost max-flow problem and thus efficiently solved. The relation between consecutive ones in *rows* matrices and flow, and, moreover, the relation of these matrices shortest and longest path problems on DAGs were first given in [27]. In [15] optimization problems on c1 matrices (on columns) are studied. The only paper that, to our knowledge, deals exactly with *MSD* is [24]. In Section 4, we will compare our heuristics in detail to the commercial OPA implementation described in [24] by applying them to the benchmark instances used in that paper.

2 Theoretical Results

To simplify the theoretical analysis of *MSD*, we restrict *MSD* instances in this section to instances where $h = 1$, that is, workforce requirements are given for a single day only, and no shifts in the collection of possible shifts span over two

days, that is, each shift starts and ends on the same day. We also assume that for the evaluation function, weights for excess and shortage are equal and are so much larger than weights for the number of shifts that the former always take precedence over the latter. This effectively gives priority to the minimization of deviation, thereby only minimizing the number of shifts for all those feasible solutions already having minimum deviation.

It is useful to describe the shifts via 0 and 1 matrices with the *consecutive ones* property. We say that a matrix A obeys the consecutive ones (c1) property if all entries in the matrix are either 0 or 1 and all the 1 in each column appear consecutively.

A column *starts* (respectively *ends*) at i if the topmost 1 entry in the column (respectively, the lowest 1 entry in the column) is in row i . A column with a single 1 entry in the i th place both starts and ends at i . The row in which a column i starts (respectively, ends) is denoted by $b(i)$ (respectively $e(i)$).

We give a formal description of MSD via c1 matrices as follows. The columns of the matrix correspond to shifts. We are given a system of inequalities: $A \cdot x \geq b$ with $x \in \mathbb{Z}^m$, $x \geq 0$, where A is an $n \times m$, c1 matrix, and b is a vector of length n of positive integers. Only x vectors meeting the above constraints are feasible. The optimization criteria is represented as follows. Let A_i be the i th row in A . Let $|x|_1$ denote the L_1 norm of x .

Input: A, b where A has the c1 property (in the columns) and the b_i are all positive.

Output: A vector $x \geq 0$ with the following properties.

1. The vector x minimizes $|Ax - b|_1$
2. Among all vectors minimizing $|Ax - b|_1$, x has minimum number of non-zero entries.

Claim. The restricted noncyclic variant of MSD where a zero deviation solution exists (namely, $Ax^* = b$ admits a solution), $h = 1$ and all shifts start and finish on the same day, is equivalent to the $UDIF$ problem.

The proof follows, followed by an explanation of how shortage and excess can be handled by a small linear adaptation of the network flow problem. This effectively allows to find the minimum (weighed) deviation from the workforce requirements (without considering minimization of the number of shifts) by solving a min-cost max-flow ($MCMF$) problem, an idea that will be reused in Section 3.

Proof. We are following here a path similar to the one in [15] in order to get this equivalence. See also, e.g., [2].

Note that in the special case when $Ax = b$ has a feasible solution, by the definition of MSD the optimum x^* satisfies $Ax^* = b$. Let $\mathcal{T}$ denote the matrix:

$$\mathcal{T} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & -1 & 0 & & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & 0 & 0 & & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix}$$

The matrix $\mathcal{T}$ is a quadratic matrix which is regular. In fact, $\mathcal{T}^{-1}$ is the upper diagonal matrix with 1 along the diagonal and above, with all other elements equal 0.

As $\mathcal{T}$ is regular the two sets of feasible vectors for $Ax = b$ and for $\mathcal{T} \cdot Ax = \mathcal{T}b$ are equal. The matrix $\mathcal{F} = \mathcal{T}A$ is a matrix with only (at most) two nonzero entries in each column: one being a 1 and the other being a -1 . In fact, all columns i in A create a column in $\mathcal{F} = \mathcal{T}A$ with exactly one -1 entry and exactly one 1 entry except for columns i with 1 in the first row (namely, so that $b(i) = 1$). These columns leave one 1 entry in row $e(i)$, namely, in the row column i ends. Call these columns the *special* columns.

The matrix $\mathcal{F}$ can be interpreted as a flow matrix (see for example [4]). Column j of the matrix is represented by an edge e_j . We assign a vertex v_i to each row i . Add an extra vertex v_0 .

An edge e_j with $\mathcal{F}_{ij} = 1$ and $\mathcal{F}_{kj} = -1$ goes out of v_k into v_i . Note that the existence of this column in $\mathcal{F}$ implies the existence in A of a column of ones starting at row $k + 1$ (and not k) and ending at row j .

In addition, for all special rows i ending at $e(i)$, we add an edge from v_0 into $v_{e(i)}$. Add an edge of capacity b_1 from s to v_0 . Let $\bar{b} = \mathcal{T}b$. The $\bar{b}$ vector determines the way all vertices (except v_0) are joined to the sink t and source s . If $\bar{b}_i > 0$ then there is an edge from v_i to t with capacity $\bar{b}_i$. Otherwise, if $\bar{b}_i < 0$, there is an edge from s to v_i with capacity $-\bar{b}_i$. Vertices with $\bar{b}_i = 0$ are not joined to the source or sink. All edges not touching the source or sink have infinite capacity.

Note that the addition of the edge from s into v_0 with capacity b_1 makes the sum of capacities of edges leaving the source equal to the sum of capacities of edges entering the sink. A saturating flow is a flow saturating all the edges entering the sink. It is easy to see that if there exists a saturating flow, then the feasible vectors for the flow problem are exactly the feasible vectors for $\mathcal{F}x = \bar{b}$. Hence, these are the same vectors feasible for the original set of equations $Ax = b$.

As we assumed that $Ax = b$ has a solution, there exists a saturating flow, namely, there is a solution saturating all the vertex-sink edges (and, in our case, all the edges leaving the source are saturated as well). Hence, the problem is transformed into the following question: Given G , find a maximum flow in G and among all maximum flows find the one that minimizes the number of proper edges carrying non-zero flow.

The resulting flow problem is in fact a *UDIF* problem. The network G is a DAG (directed acyclic graph). This clearly holds true as all edges go from v_i to

v_j with $j > i$. In addition, all capacities on edges not touching the sink or source are infinite (see the above construction).

On the other hand, given a *UDIF* instance with a saturating flow (namely, where one can find a flow function saturating all the edges entering the sink) it is possible to find an inverse function that maps it to an *MSD* instance. The *MSD* instance is described as follows.

Assume that the v_i are ordered in increasing topological order. Given the DAG G , the corresponding matrix $\mathcal{F}$ is defined by taking the edge-vertices incidence matrix of G . As it turns out, we can find a c1 matrix A so that $\mathcal{T}A = \mathcal{F}$. Indeed, for any column j with non-zeros in rows q, p with $q < p$, necessarily, $\mathcal{F}_{qj} = -1$ and $\mathcal{F}_{pj} = 1$ (if there is a column j that does not contain an $\mathcal{F}_{qj} = -1$, set $q = 0$). Hence, add to A the c1 column with 1 from rows $q + 1$ to p .

We note that the restriction of the existence of a flow saturating the flow along edges entering t is not essential. It is easy to guarantee this as follows. Add a new vertex u to the network and an edge (s, u) of capacity $\sum_{(v,t)} c(v, t) - f^*$ (where f^* is the maximum flow value). By definition, the edge (s, u) has cost 0. Add a directed edge from u to every source v . This makes a saturating flow possible, at the increase of only 1 in the cost.

It follows that in the restricted case when $Ax = b$ has feasible solutions the *MSD* problem is equivalent to *UDIF*. $\square$

To understand how this can be used to also find solutions to *MSD* instances where no zero deviation solution exists, we need to explain how to find a vector x so that $Ax \geq b$ and $|Ax - b|_1$ is minimum. When $Ax = b$ does not have a solution, we introduce n dummy variables y_i . The i th inequality is replaced by $A_i x - y_i = b_i$, namely, y_i is set to the difference between $A_i x$ and b_i (and $y_i \geq 0$). Let $-I$ be the negative identity matrix, namely, the matrix with all zeros except -1 in the diagonal entries. Let $(A; -I)$ be the A matrix with $-I$ to its right and let $(x; y)$ be the column of x followed by the y variables. The above system of inequalities is represented by $(A; -I)(x; y) = b$. Multiplying the inequality by $\mathcal{T}$ (where $\mathcal{T}$ is the 0, 1 and -1 matrix defined above) gives $(\mathcal{F}; -\mathcal{T})(x; y) = \mathcal{T}b = \bar{b}$. The matrix $(\mathcal{F}; -\mathcal{T})$ is a flow matrix. Its corresponding graph is the graph of $\mathcal{F}$ with the addition of an infinite capacity edge from v_i into v_{i-1} ($i = 1, \dots, n$). Call these edges the y edges. The edges originally in G are called the x edges. The sum $\sum_i y_i$ clearly represents the excess L_1 norm $|Ax - b|_1$. Hence, we give a cost $C(e) = 1$ to each edge corresponding to a y_i . We look for a maximum flow minimizing $\sum_i C(e)f(e)$, namely, a min-cost max-flow solution. As we may assume w.l.o.g. that all time intervals $[t_i, t_{i+1})$ ($i = 1, \dots, n$) have equal length, this gives the minimum possible excess. Shortage can be handled in a similar way.

We next show that unless $P = NP$, there is some constant $0 < c < 1$ such that approximating *UDIF* within a $c \ln n$ -ratio is NP-hard.

Since the case of zero excess *MSD* is equivalent to *UDIF* (see Claim 2), similar hardness results follow for this problem as well.

Theorem 1. *There is a constant $c < 1$ so that approximating the UDIF problem within $c \ln n$ is NP-hard.*

Proof. We prove a hardness reduction for UDIF under the assumption $P \neq NP$. We use a reduction from Set-Cover. We need a somewhat different proof than [18] to account for the extra restriction imposed by UDIF. For our purposes it is convenient to formulate the set cover problem as follows. The set cover instance is an undirected bipartite graph $\mathcal{B}(V_1, V_2, A)$ with edges only crossing between V_1 and V_2 . We may assume that $|V_1| = |V_2| = n$. We look for a minimum sized set $S \subseteq V_1$ so that $N(S) = V_2$ (namely, every vertex in V_2 has a neighbor in S). If $N(S) = V_2$ we say that S covers V_2 . We may assume that the given instance has a solution. The following is proven in [26].

Theorem 2. *There is a constant $c < 1$ so that approximating Set-Cover within $c \ln n$ is NP-hard.*

We prove a similar result for UDIF and thus for MSD.

Let $\mathcal{B}(V_1, V_2, \mathcal{E})$ be the instance of the set cover problem at hand so that $|V_1| = |V_2| = n$. Add a source s and a sink t . Connect s to all the vertices of V_2 with capacity one edges. Direct all the edges of $\mathcal{B}$ from V_2 to V_1 . Now, create n^2 copies V_1^i of V_1 and for convenience denote $V_1 = V_1^0$. For each $i \in \{0, \dots, n^2 - 1\}$, connect in a directed edge the copy $v_1^i \in V_1^i$ of each $v_1 \in V_1$ to the copy $v_1^{i+1} \in V_1^{i+1}$ of v_1 in V_1^{i+1} . Hence, a perfect matching is formed between contiguous V_1^i via the copies of the $v_1 \in V_1$ vertices. The vertices of $V_1^{n^2}$ are all connected to t via edges of capacity n . Note that by definition, all other edges (which are edges touching neither the source nor the sink) have infinite capacity.

It is straightforward to see that the resulting graph is a DAG and that the graph admits a flow saturating the source edges, and can be made to saturate the sink edges as described before.

We now inspect the properties of a “good” solution. Let S be the set of vertices $S \subseteq V_1$ so that for every vertex $v_2 \in V_2$ there exists a vertex $s \in S$ such that edge (v_2, s) carries positive flow.

Note that for every $v_2 \in V_2$ there must be such an edge for otherwise the flow is not optimal. Further note that the flow units entering S must be carried throughout the copies of S in all of the V_1^i sets $i \geq 1$ using the matching edges as this is the only way to deliver the flow into t . Hence, the number of proper edges in the solution is exactly $n^2 \cdot |S| + n$. The n term comes from the n edges touching the vertices of V_2 .

Further, note that S must be a set cover of V_2 in the original graph $\mathcal{B}$. Indeed, every vertex v_2 must have a neighbor in S . Finally, note that it is indeed possible to get a solution with $n^2 \cdot s^* + n$ edges where s^* is the size of the minimum set cover using an optimum set cover S^* as described above. Since all the matching edges have infinite capacities, it is possible to deliver to t the n units of flow regardless of how the cover S is chosen. The following properties end the proof: The number of vertices n' in the new graph is $O(n^3)$. In addition, the additive term n is negligible for large enough n in comparison to $n^2 \cdot |S|$ where S is the chosen set cover. Hence, the result follows for $c < 1/3 < 1$. $\square$

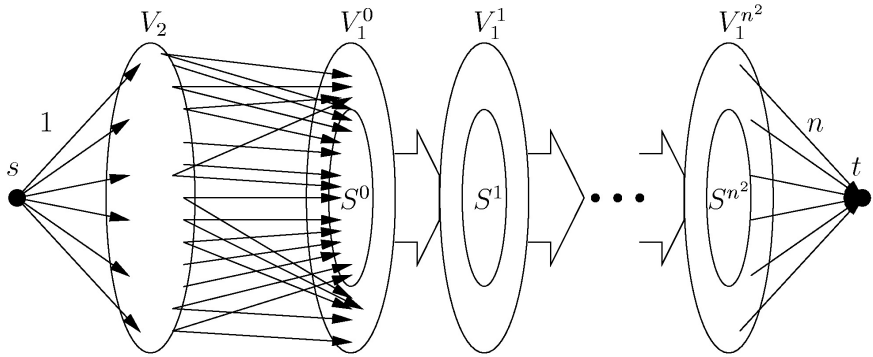


Fig. 1. Schematic illustration of the reduction from the Set-Cover problem to the UDIF problem.

3 Practical Heuristics

We implemented three practical heuristics. The first, H1, is a local search procedure based on interleaving different neighborhood definitions. The second, H2, is a new greedy heuristic that uses a min-cost max-flow (*MCMF*) subroutine, inspired by the relation between the *MSD* and *MECF* problems. The third solver, H3, consists in a serial combination of H1 and H2.

Our first solver, H1, is fully based on the local search paradigm [1]. Differently from Musliu *et al.* [24], that use tabu search as well, we use three neighborhood relations selectively in various phases of the search, rather than exploring the overall neighborhood at each iteration. The reason for using limited neighborhood relations is not related to the saving of computational time, which could be obtained in other ways, for example by clever ordering of promising moves. The main reason, instead, is the introduction of a certain degree of *diversification* in the search.

Our second solver, H2, is based on a simple greedy heuristic that uses a polynomial min-cost max-flow subroutine *MCMF*(), based on the equivalence of the (non-cyclic) *MSD* problem to *UDIF*, a special case of the *MECF* problem for which no efficient algorithm is known (see Section 2), and the relationship of the latter with the *MCMF* problem for which efficient algorithms are known. It is based on the observation that the *MCMF* subroutine can easily compute the optimal staffing with minimum (weighted) deviation when slack edges have associated costs corresponding, respectively, to the weights of shortage and excess. Note that it is not able to simultaneously minimize the number of shifts that are used. After some preprocessing to account for cyclicity, the greedy heuristic then removes all shifts that did not contribute to the *MSD* instance corresponding to the current flow computed with *MCMF*(). It randomly chooses one shift (without repetitions) and tests whether removal of this shift still allows the *MCMF*() to find a solution with the same deviation. If this is the case, that shift is removed and not considered anymore, otherwise it is left in the set of shifts used to build

the network flow instances, but will not be considered for removal again. Finally, when no shifts can be removed anymore without increasing the deviation, a final simple postprocessing step is made to restore cyclicity.

4 Computational Results

All experiments were made on sets of instances that are available in self-describing text files from

<http://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html>.

A detailed description of the random instance generator used to construct them can be found in [24]. We remark that our solvers produce results much better than the solver of OPA. In fact, H1 always finds the best solution, H2 in 21 cases, and H3 in 29 cases, whereas OPA finds the best solution only in 17 instances. H1, although it finds the best solution, is always much slower than H2, and generally slower than H3 as well. To show how heuristics scale up, we analyzed the performance for our solvers within 10 seconds time based on the size of the problems. These experiments show that for short runs H1 is clearly inferior to H2 and H3, which are comparable. The above experiments show that H1 is superior in reaching the best known solution, but it requires more time than H2. On the time-limited experiments H2 is clearly superior to H1. The solver H3 has the good qualities of both, and therefore it can be considered the best general-purpose solver. Further tests on more examples confirm these trends and are omitted for brevity.

Acknowledgments. This work was supported by Austrian Science Fund Project No. Z29-N04.

References

1. Emile Aarts and Jan Karl Lenstra, editors. *Local Search in Combinatorial Optimization*. Wiley, 1997.
2. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.
3. K. Arata, S. Iwata, K. Makino, and S. Fujishige. Source location: Locating sources to meet flow demands in undirected networks. In *SWAT*, 2000.
4. J. Bar-Ilan, G. Kortsarz, and D. Peleg. Generalized submodular cover problems and applications. In *The Israeli Symposium on the Theory of Computing*, pages 110–118, 1996. Also in *Theoretical Computer Science*, to appear.
5. J.J. Bartholdi, J.B. Orlin, and H.D. Ratliff. Cyclic scheduling via integer programs with circular ones. *Operations Research*, 28:110–118, 1980.
6. M. Bellare. Interactive proofs and approximation: reduction from two provers in one round. In *The second Israeli Symposium on the Theory of Computing*, pages 266–274, 1993.
7. R.D. Carr, L.K. Fleischer, V.J. Leung, and C.A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proc. of the 11th ACM/SIAM Symposium on Discrete Algorithms*, 2000.

8. L. Equi, G. Gallo, S. Marziale, and A. Weintraub. A combined transportation and scheduling problem. *European Journal of Operational Research*, 97(1):94–104, 1997.
9. Guy Even, Guy Kortsarz, and Wolfgang Slany. On network design problems: Fixed cost flows and the covering steiner problem. In *8th Scandinavian Workshop on Algorithm Theory (SWAT)*, LNCS 2368, pages 318–329, 2002.
10. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co., 1979.
11. N. Garg, M. Yannakakis, and V.V. Vazirani. Approximating max-flow min-(multi)cut theorems and their applications. *Siam J. on Computing*, 25:235–251, 1996.
12. N. Garg, M. Yannakakis, and V.V. Vazirani. Primal-dual approximation algorithms for integral flow and multicuts in trees. *Algorithmica*, 18:3–20, 1997.
13. Johannes Gärtner, Nysret Musliu, and Wolfgang Slany. Rota: a research project on algorithms for workforce scheduling and shift design optimization. *AI Communications: The European Journal on Artificial Intelligence*, 14(2):83–92, 2001.
14. M. Goethe-Lundgren and T. Larsson. A set covering reformulation of the pure fixed charge transportation problem. *Discrete Appl. Math.*, 48(3):245–259, 1994.
15. D. Hochbaum. Optimization over consecutive 1's and circular 1's constraints. Unpublished manuscript, 2000.
16. D.S. Hochbaum and A. Segev. Analysis of a flow problem with fixed charges. *Networks*, 19(3):291–312, 1989.
17. D. Kim and P.M. Pardalos. A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Oper. Res. Lett.*, 24(4):195–203, 1999.
18. S.O. Krumke, H. Noltemeier, S. Schwarz, H.-C. Wirth, and R. Ravi. Flow improvement and network flows with fixed costs. In *OR-98*, Zürich, 1998.
19. G. Laporte. The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017, 1999.
20. H.C. Lau. Combinatorial approaches for hard problems in manpower scheduling. *J. Oper. Res. Soc. Japan*, 39(1):88–98, 1996.
21. C.E. Leiserson and J.B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.
22. T.L. Magnanti and R.T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18:1–55, 1984.
23. Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.
24. Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. Local search for shift design. *European Journal of Operational Research* (to appear).
<http://www.dbai.tuwien.ac.at/proj/Rota/DBAI-TR-2001-45.ps>.
25. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.
26. R. Raz and S. Safra. A sub constant error probability low degree test, and a sub constant error probability PCP characterization of NP. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 475–484, 1997.
27. A.F. Veinott and H.M. Wagner. Optimal capacity scheduling: Parts i and ii. *Operation Research*, 10:518–547, 1962.

Loglog Counting of Large Cardinalities

(Extended Abstract)

Marianne Durand and Philippe Flajolet

Algorithms Project, INRIA–Rocquencourt, F78153 Le Chesnay (France)

Abstract. Using an auxiliary memory smaller than the size of this abstract, the LOGLOG algorithm makes it possible to estimate in a single pass and within a few percents the number of different words in the whole of Shakespeare’s works. In general the LOGLOG algorithm makes use of m “small bytes” of auxiliary memory in order to estimate in a single pass the number of distinct elements (the “*cardinality*”) in a file, and it does so with an accuracy that is of the order of $1/\sqrt{m}$. The “small bytes” to be used in order to count cardinalities till $N_{\max}$ comprise about $\log \log N_{\max}$ bits, so that cardinalities well in the range of billions can be determined using one or two kilobytes of memory only. The basic version of the LOGLOG algorithm is validated by a complete analysis. An optimized version, super-LOGLOG, is also engineered and tested on real-life data. The algorithm parallelizes optimally.

1 Introduction

The problem addressed in this note is that of determining the number of *distinct* elements, also called the *cardinality*, of a large file. This problem arises in several areas of data-mining, database query optimization, and the analysis of traffic in routers. In such contexts, the data may be either too large to fit at once in core memory or even too massive to be stored, being a huge continuous flow of data packets. For instance, Estan *et al.* [3] report traces of packet headers, produced at a rate of 0.5GB per hour of compressed data (!), which were collected while trying to trace a “worm” (Code Red, August 1 to 12, 2001), and on which it was necessary to count the number of *distinct* sources passing through the link. We propose here the LOGLOG algorithm that estimates cardinalities using only a very small amount of auxiliary memory, namely m memory units, where a memory unit, a “*small byte*”, comprises close to $\log \log N_{\max}$ bits, with $N_{\max}$ an *a priori* upperbound on cardinalities. The estimate is (in the sense of mean values) asymptotically *unbiased*; the relative accuracy of the estimate (measured by a standard deviation) is close to $1.05/\sqrt{m}$ for our best version of the algorithm, Super-LOGLOG. For instance, estimating cardinalities till $N_{\max} = 2^{27}$ (a hundred million different records) can be achieved with $m = 2048$ memory units of 5 bits each, which corresponds to 1.28 kilobytes of auxiliary storage in total, the error observed being typically less than 2.5%. Since the algorithm operates incrementally and in a single pass it can be applied to data flows for which it provides on-line estimates available at any given time. Advantage can be taken

of the low memory consumption in order to gather *simultaneously* a *very large number of statistics* on huge heterogeneous data sets. The LOGLOG algorithm can also be *fully distributed* or *parallelized*, with *optimum* speed-up and *minimal* interprocess communication. Finally, an embedded hardware design would involve strictly minimal resources.

Motivations. A traditional application of cardinality estimates is database query optimization. There, a complex query typically involves a variety of set-theoretic operations as well as projections, joints, and so on. In this context, knowing “for free” cardinalities of associated sets provides a valuable guide for selecting an efficient processing strategy best suited to the data at hand. Even a problem as simple as merging two large files with duplicates can be treated by various combinations of sorting, straight merging, and filtering out duplicates (in one or both of the files); the cost function of each possible strategy is then determined by the number of records as well as by the cardinality of each file. Probabilistic estimation algorithms also find a use in large data recording and warehousing environments. There, the goal is to provide an approximate response in time that is orders-of-magnitude less than what computing an exact answer would require: see the description of the AQUA Project by Gibbons *et al.* in [8].

The analysis of traffic in routers, as already mentioned, benefits greatly of cardinality estimators—this is lucidly exposed by Estan *et al.* in [2,3]. Certain types of attacks (“denial of service” and “port scans”) are betrayed by alarmingly high counts of certain characteristic events in routers. In such situations, there is usually not enough resource available to store and search on-line the very large number of events that take place even in a relatively small time window.

Probabilistic counting algorithms can also be used within other algorithms whenever the final answer is the cardinality of a large set and a small tolerance on the quality of the answer is acceptable. Palmer *et al.* [13] describe the use of such algorithms in an extensive connectivity analysis of the internet topology. For instance, one of the tasks needed there is to determine, for each distance h , the number of pairs of nodes that are at distance at most h in the internet graph. Since the graph studied by [13] has close to 300,000 nodes, the number of pairs to be considered is well over 10^{10} , upon which costly list operations must be performed by exact algorithms. In contrast an algorithm that would be, in the abstract, suboptimal can be coupled with adapted probabilistic counting techniques and still provide reliable estimates. In this way, the authors of [13] were able to extract extensive metric information on the internet graph by keeping a reduced collection of data that reside in core memory. They report a reduction in run-time by a factor of more than 400.

Algorithms. The LOGLOG algorithm is probabilistic. Like in many similar algorithms, the first idea is to appeal to a hashing function in order to randomize data and bring them to a form that resembles random (uniform, independent) binary data. It is this hashed data set that is distilled into cardinality estimates by the algorithm. Various algorithms perform various tests on the hashed data set, then compare “*observables*” to what probabilistic analysis predicts, and finally “deduce” a plausible value of the parameter of interest. In the case of

```
ghfffghfghgghggggghgheehfhhghgghghhfgffffhhhiigfhhffgfiihfhhh
igigighfghfffghigihghigfhhgeeghggghhghhfhidiigihighihehhffgg
hfgighigffghdieghhggghhfhghfiieffghghihifgggffihgihfggighgiiif
fjfgfjhhjiifhjgehgghfhhfhjhiggghghihigghhihihgiighgfhlgjfgjjjmf1
```

The LOGLOG Algorithm with $m = 256$ condenses the whole of Shakespeare’s works to a table of 256 “small bytes” of 4 bits each. The estimate of the number of distinct words is here $n^\circ = 30897$ (true answer: $n = 28239$), i.e., a relative error of +9.4%.

LOGLOG counting, the observable should *only* be linked to cardinality, and hence be totally independent of the nature of replications and the ordering of data present in the file, on which no information at all is available. (Depending on context, collisions due to hashing can either be neglected or their effect can be estimated and corrected.)

Whang, Zanden, and Taylor [16] have developed *Linear Counting*, which distributes (hashed) values into buckets and only keeps a bitmap indicating which buckets are hit. Then observing the number of hits in the table leads to an estimate of cardinality. Since the number of buckets should not be much smaller than the cardinalities to be estimated (say, $\geq N_{\max}/10$), the algorithm has space complexity that is $O(N_{\max})$ (typically, $N_{\max}/10$ bits of storage). The linear space is a drawback whenever large cardinalities, multiple counts, or limited hardware are the rule. Estan, Varghese, and Fisk [3] have devised a multiscale version of this principle, where a hierarchical collection of small windows on the bitmap is kept. From simulation data, their *Multiresolution Bitmap* algorithm appears to be about 20% more accurate than Probabilistic Counting (discussed below) when the same amount of memory is used. The best algorithm of [3] for flows in routers, *Adaptive Bitmap*, is reported to be about 3 times more efficient than either Probabilistic Counting or Multiresolution Bitmap, but it has the disadvantage of not being *universal*, as it makes definite statistical assumptions (“stationarity”) regarding the data input to the algorithm. (We recommend the thorough engineering discussion of [3].)

Closer to us is the *Probabilistic Counting* algorithm of Flajolet and Martin [7]. This uses a certain observable that has excellent statistical properties but is relatively costly to maintain in terms of storage. Indeed, Probabilistic Counting estimates cardinalities with an error close to $0.78/\sqrt{m}$ given a table of m “words”, each of size about $\log_2 N_{\max}$.

Yet another possible idea is sampling. One may use any filter on hashed values with selectivity $p \ll 1$, store exactly and without duplicates the data items filtered and return as estimate $1/p$ times the corresponding cardinality. Wegner’s *Adaptive Sampling* (described and analyzed in [5]) is an elegant way to maintain dynamically varying values of p . For m “words” of memory (where here “word” refers to the space needed by a data item), the accuracy is about $1.20/\sqrt{m}$, which is about 50% less efficient than Probabilistic Counting.

An insightful complexity-theoretic discussion of approximate counting is provided by Alon, Matias, and Szegedy in [1]. The authors discuss a class of “frequency-moments” statistics which includes ours (as their F_0 statistics). Our

LOGLOG Algorithm has principles that evoke some of those found in the intersection of [1] and the earlier [7], but contrary to [1], we develop here a complete eminently *practical* algorithmic solution and provide a very precise analysis, including bias correction, error and risk evaluation, as well as complete dimensioning rules.

We estimate that our LOGLOG algorithm outperforms the earlier Probabilistic Counting algorithm and the similarly performing Multiresolution Bitmap of [3] by a factor of 3 at least as it replaces “words” (of 16 to 32 bits) by “*small bytes*” of typically 5 bits each, while being based on an observable that has only slightly higher dispersion than the other two algorithms—this is expressed by our two formulæ $1.30/\sqrt{m}$ (LOGLOG) and $1.05/\sqrt{m}$ (super-LOGLOG). This places our algorithm in the same category as Adaptive Bitmap of [3]. However, compared to Adaptive Bitmap, the LOGLOG algorithm has the great advantage of being *universal* as it makes no assumptions on the statistical regularity of data. We thus believe LOGLOG and its improved version Super-LOGLOG to be the best general-purpose algorithmic solution currently known to the problem of estimating large cardinalities.

Note. The following related references were kindly suggested by a referee: Cormode *et al.*, in VLDB–2002 (a new counting method based on stable laws) and Bar-Yossef *et al.*, SODA–2002 (a new application to counting triangles in graphs).

2 The Basic LOGLOG Algorithm

In computing practice, one deals with a *multiset* of data items, each belonging to a discrete universe $\mathcal{U}$. For instance, in the case of natural text, $\mathcal{U}$ may be the set of all alphabetic strings of length ≤ 28 (‘antidisestablishmentarianism’), double floats represented on 64 bits, and so on. A multiset $\mathfrak{M}$ of elements of $\mathcal{U}$ is given and the problem is to estimate its cardinality, that is, the number of *distinct* elements it comprises. Here is the principle of the basic LOGLOG algorithm.

```

Algorithm LOGLOG( $\mathfrak{M}$ : Multiset of hashed values;  $m \equiv 2^k$ )
Initialize  $M^{(1)}, \dots, M^{(m)}$  to 0;
let  $\rho(y)$  be the rank of first 1-bit from the left in  $y$ ;
  for  $x = b_1b_2 \dots \in \mathfrak{M}$  do
    set  $j := \langle b_1 \dots b_k \rangle_2$  (value of first  $k$  bits in base 2)
    set  $M^{(j)} := \max(M^{(j)}, \rho(b_{k+1}b_{k+2} \dots))$ ;
return  $E := \alpha_m m 2^{\frac{1}{m} \sum_j M^{(j)}}$  as cardinality estimate.

```

We assume throughout that a *hash function*, h , is available that transforms elements of $\mathcal{U}$ into sufficiently long binary strings, in such a way that bits composing the hashed value closely resemble random uniform independent bits. This pragmatic attitude¹ is justified by Knuth who writes in [10]: “*It is theoretically*

¹ The more theoretically inclined reader may prefer to draw h at random from a family of universal hash functions; see, e.g., the general discussion in [12] and the specific [1].

impossible to define a hash function that creates random data from non-random data in actual files. But in practice it is not difficult to produce a pretty good imitation of random data.” Given this, we formalize our basic problem as follows.

Take $\mathcal{U} = \{0, 1\}^\infty$ as the universe of data endowed with the uniform (product) probability distribution. An *ideal multiset* $\mathfrak{M}$ of cardinality n is a random object that is produced by first drawing an n -sequence independently at random from $\mathcal{U}$, then replicating elements in an arbitrary way, and finally, applying an arbitrary permutation.

The user is provided with the (extremely large) ideal multiset $\mathfrak{M}$ and its goal is to estimate the (unknown to him) value of n at a small computational cost. No information is available, hence no statistical assumption can be made, regarding the behaviour of the replicator-shuffler daemon.

(The fact that we consider infinite data is a convenient abstraction at this stage; we discuss its effect, together with needed adjustments, in Section 5 below.)

The basic idea consists in scanning $\mathfrak{M}$ and observing the patterns of the form 0^*1 that occur at the beginning of (hashed) records. For a string $x \in \{0, 1\}^\infty$, let $\rho(x)$ denote the position of its first 1-bit. Thus $\rho(1 \cdots) = 1$, $\rho(001 \cdots) = 3$, etc. Clearly, we expect about $n/2^k$ amongst the distinct elements of $\mathfrak{M}$ to have a ρ -value equal to k . In other words, the quantity,

$$R(\mathfrak{M}) := \max_{x \in \mathfrak{M}} \rho(x),$$

can reasonably be hoped to provide a rough indication on the value of $\log_2 n$. It is an “observable” in the sense above since it is totally independent of the order and the replication structure of the multiset $\mathfrak{M}$. In fact, in probabilistic terms, the quantity R is precisely distributed in the same way as 1 plus the maximum of n independent geometric variables of parameter $\frac{1}{2}$. This is an extensively researched subject; see, e.g., [14]. It turns out that R estimates $\log_2 n$ with an additive bias of 1.33 and a standard deviation of 1.87. Thus, in a sense, the observed value of R estimates “logarithmically” n within ± 1.87 binary orders of magnitude. Notice however that the expectation of 2^R is infinite so that 2^R cannot in fact be used to estimate n .

The next idea consists in separating elements into m groups also called “buckets”, where m is a design parameter. With $m = 2^k$, this is easily done by using the first k bits of x as representing in binary the index of a bucket. One can then compute the parameter R on each bucket, after discarding the first k bits. If $M^{(j)}$ is the (random) value of parameter R on bucket number j , then the arithmetic mean $\frac{1}{m} \sum_{j=1}^m M^{(j)}$, can legitimately be expected to approximate $\log_2(n/m)$ plus an additive bias. The estimate of n returned by the LOGLOG algorithm is accordingly

$$E := \alpha_m m 2^{\frac{1}{m} \sum M^{(j)}}. \quad (1)$$

The constant α_m comes out of our later analysis as $\alpha_m := \left(\Gamma(-1/m) \frac{1-2^{1/m}}{\log 2} \right)^{-m}$, where $\Gamma(s) := \frac{1}{s} \int_0^\infty e^{-t} t^s dt$. It precisely corrects the systematic bias of the raw arithmetic mean in the asymptotic limit. One may also hope for a greater concentration of the estimates, hence better accuracy, to result from averaging over $m \gg 1$ values. The main characteristics

of the algorithm are summarized below in Theorem 1. The letters $\mathbb{E}, \mathbb{V}$ denote expectation and variance, and the subscript n indicates the cardinality of the underlying random multiset.

Theorem 1. *Consider the basic LOGLOG algorithm applied to an ideal multiset of (unknown) cardinality n and let E be the estimated value of cardinality returned by the algorithm.*

(i) *The estimate E is asymptotically unbiased in the sense that, as $n \rightarrow \infty$,*

$$\frac{1}{n} \mathbb{E}_n(E) = 1 + \theta_{1,n} + o(1), \quad \text{where } |\theta_{1,n}| < 10^{-6}.$$

(ii) *The standard error defined as $\frac{1}{n} \sqrt{\mathbb{V}_n(E)}$ satisfies as $n \rightarrow \infty$,*

$$\frac{1}{n} \sqrt{\mathbb{V}_n(E)} = \frac{\beta_m}{\sqrt{m}} + \theta_{2,n} + o(1), \quad \text{where } |\theta_{2,n}| < 10^{-6}.$$

One has: $\beta_{128} \doteq 1.30540$, $\beta_\infty = \sqrt{\frac{1}{12} \log^2 2 + \frac{1}{6} \pi^2} \doteq 1.29806$.

In summary, apart from completely negligible fluctuations whose amplitude is less than 10^{-6} , the algorithm provides asymptotically a valid estimator of n . The standard error, which measures in a mean-quadratic sense and in proportion to n the deviations to be expected, is closely approximated by the formula²

$$\text{Standard error} \approx \frac{1.30}{\sqrt{m}}.$$

For instance, $m = 256$ and $m = 1024$ give a standard error of 8% and 4% respectively. (These figures are compatible with what was obtained on the Shakespeare data.) Observe also that $\alpha_m \sim \alpha_\infty - (2\pi^2 + \log^2 2)/(48m)$, where $\alpha_\infty = e^{-\gamma} \sqrt{2}/2 \doteq 0.39701$ (γ is Euler's constant), so that, in practical implementations, α_m can be replaced by α_∞ without much detectable bias as soon as $m \geq 64$.

The proof of Theorem 1 will occupy the whole of the next section.

3 The Basic Analysis

Throughout this note, the *unknown* number of distinct values in the data set is denoted by n . The LOGLOG algorithm provides an estimator, E , of n . We first provide formulæ for the expectation and variance of E . Asymptotic analysis is performed next: The Poissonization paragraph introduces the Poisson model where n is allowed to vary according to a Poisson law, while the Depoissonization paragraph shows the Poisson model to be asymptotically equivalent to the “fixed- n ” model that we need. The expected value of the estimator is found to be asymptotically n , up to minute fluctuations. This establishes the asymptotically unbiased character of the algorithm as asserted in (i) of Theorem 1. The standard deviation of the estimator is also proved to be of the order of n with the proportionality coefficient providing the value of the standard error, hence the accuracy of the algorithm, as asserted in (ii) of Theorem 1.

² We use ‘ $\sim$ ’ to denote asymptotic expansions in the usual mathematical sense and reserve the informal ‘ $\approx$ ’ for “approximately equal”.

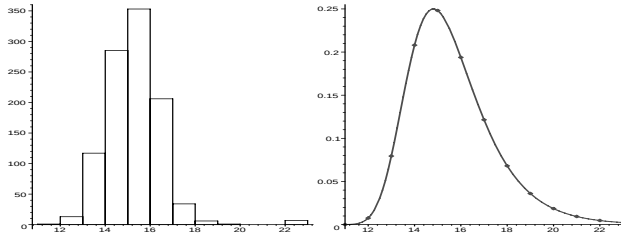


Fig. 1. The distribution of observed register values for the Pi file, $n \approx 2 \cdot 10^7$ with $m = 1024$ [left]; the distribution $\mathbb{P}_\nu(M = k)$ of a register M , for $\nu = 2 \cdot 10^4$ [right].

We start by examining what happens in a bucket that receives ν elements (Figure 1). The random variable M is, we recall, the maximum of ν random variables that are independent and geometrically distributed according to $\mathbb{P}(Y \geq k) = \frac{1}{2^{k-1}}$. Consequently, the probability distribution of M is characterized by $\mathbb{P}_\nu(M \leq k) = (1 - \frac{1}{2^k})^\nu$, so that $\mathbb{P}_\nu(M = k) = (1 - \frac{1}{2^k})^\nu - (1 - \frac{1}{2^{k-1}})^\nu$. The bivariate (exponential) generating function of this family of probability distributions as ν varies is then

$$G(z, u) := \sum_{\nu, k} \mathbb{P}_\nu(M = k) u^k \frac{z^\nu}{\nu!} = \sum_k u^k \left(e^{z(1-1/2^k)} - e^{z(1-1/2^{k-1})} \right), \quad (2)$$

as shown by a simple calculation. The starting point of the analysis is an expression in terms of G of the mean and variance of $Z := E/\alpha_m \equiv m 2^{\frac{1}{m}} \sum_j M^{(j)}$, which is the unnormalized version of the estimator E . With the expression $[z^n]f(z)$ representing the coefficient of z^n in the power series $f(z)$, we state:

Lemma 1. *The expected value and variance of the unnormalized estimator Z are $\mathbb{E}_n(Z) = mn![z^n]G(\frac{z}{m}, 2^{1/m})^m$, and*

$$\mathbb{V}_n(Z) = m^2 n![z^n] \left(G\left(\frac{z}{m}, 2^{2/m}\right) \right)^m - \left(mn![z^n]G\left(\frac{z}{m}, 2^{1/m}\right)^m \right)^2$$

Proof. The multinomial convolution relations corresponding to m th powers of generating functions imply that $n![z^n]G(z/m, u)^m$ is the probability generating function of $\sum_j M^{(j)}$. (The multinomials enumerate all ways of distributing elements amongst buckets.) The expressions for the first and second moment of Z are obtained from there by substituting $u \mapsto 2^{1/m}$ and $u \mapsto 2^{2/m}$.

Proving Theorem 1 is reduced to estimating asymptotically these quantities.

Poissonization. We “poissonize” the problem of computing the expected value and the variance. In this way, calculations take advantage of powerful properties of the Mellin transform. The Poisson law of rate λ is the law of a random variable X such that $\mathbb{P}(X = \ell) = e^{-\lambda} \frac{\lambda^\ell}{\ell!}$. Given a class $\mathcal{M}_s$ of probabilistic models indexed by integers s , poissonizing means considering the “supermodel” where model $\mathcal{M}_s$ is chosen according to a Poisson law of rate λ . Since the poisson model of a large parameter λ is predominantly a mixture of models $\mathcal{M}_s$ with s near λ (the Poisson law is “concentrated” near its mean), one can expect

properties of the fixed- n model $\mathcal{M}_n$ to be reflected by corresponding properties of the Poisson model taken with rate $\lambda = n$.

A useful feature is that expressions of moments and probabilities under the Poisson model are closely related to exponential generating functions of the fixed- n models. This owes to the fact that if $f(z) = \sum_n f_n z^n / n!$ is the exponential generating function of expectations of a parameter, then the quantity $e^{-\lambda} f(\lambda) = \sum_n f_n e^{-\lambda} \frac{\lambda^n}{n!}$ gives the corresponding expectation under the Poisson model. In this way, one sees that the quantities $\mathcal{E}_n = mG\left(\frac{n}{m}, 2^{1/m}\right)^m e^{-n}$ and $\mathcal{V}_n = m^2 G\left(\frac{n}{m}, 2^{2/m}\right)^m e^{-n} - \left(mG\left(\frac{n}{m}, 2^{1/m}\right)^m e^{-n}\right)^2$ are respectively the mean and variance of Z when the cardinality of the underlying multiset obeys a Poisson law of rate $\lambda = n$.

Lemma 2. *The Poisson mean and variance $\mathcal{E}_n$ and $\mathcal{V}_n$ satisfy as $n \rightarrow \infty$:*

$$\begin{aligned}\mathcal{E}_n &\sim \left[\left(\Gamma(-1/m) \frac{1 - 2^{1/m}}{\log 2} \right)^m + \epsilon_n \right] \cdot n \\ \mathcal{V}_n &\sim \left[\left(\Gamma(-2/m) \frac{1 - 2^{2/m}}{\log 2} \right)^m - \left(\Gamma(-1/m) \frac{1 - 2^{1/m}}{\log 2} \right)^{2m} + \eta_n \right] \cdot n^2.\end{aligned}$$

where $|\epsilon_n|$ and $|\eta_n|$ are bounded by 10^{-6} .

The proof crucially relies on the Mellin transform [6].

Depoissonization. Finally, the asymptotic forms of the first two moments of the LOGLOG estimator can be transferred back from the Poisson model to the fixed- n model that underlies Theorem 1. The process involved is known as “depoissonization”. Various options are discussed in Chapter 10 of Szpankowski’s book [15]. We choose the method called “analytic depoissonization” by Jacquet and Szpankowski, whose underlying engine is the saddle point method applied to Cauchy integrals; see [9,15]. In essence, the *values* of an exponential generating function at large arguments are closely related to the asymptotic form of its *coefficients* provided the generating function decays fast enough away from the positive real axis in the complex plane. The complete proof is omitted.

Lemma 3. *The first two moments of the LOGLOG estimator are asymptotically equivalent under the Poisson and fixed- n model: $\mathbb{E}_n(Z) \sim \mathcal{E}_n$, and $\mathbb{V}_n(Z) \sim \mathcal{V}_n$.* Lemmas 2 and 3 together prove Theorem 1. Easy numerical calculations and straight asymptotic analysis of β_m conclude the evaluations stated there.

4 Space Requirements

Now that the correctness—the absence of bias as well as accuracy—of the basic LOGLOG algorithm has been established, there remains to see that it performs as promised and only consumes $O(\log \log n)$ bits of storage if counts till n are needed³.

³ A counting algorithm exhibiting a log-log feature in a different context is Morris’s *Approximate Counting* [11] analyzed in [4].

In its abstract form of Section 1, the LOGLOG algorithm operates with potentially unbounded integer registers and it consumes m of these. What we call an ℓ -restricted algorithm is one in which each of the $M^{(j)}$ registers is made of ℓ bits, that is, it can store any integer between 0 and $2^\ell - 1$. We state a shallow result only meant to phrase mathematically the log-log property of the basic space complexity:

Theorem 2. *Let $\omega(n)$ be a function that tends to infinity arbitrarily slowly and consider the function $\ell(n) = \log_2 \log_2 \left(\frac{n}{m}\right) + \omega(n)$. Then, the $\ell(n)$ -restricted algorithm and the LOGLOG algorithm provide the same output with probability tending to 1 as n tends to infinity.*

The auxiliary tables maintained by the algorithm then comprise m “small bytes”, each of size $\ell(n)$. In other words, the total space required by the algorithm in order to count till n is $m \log_2 \log_2 \left(\frac{n}{m}\right) (1 + o(1))$. The hashing function needs to hash values from the original data universe onto exactly $2^{\ell(n)} + \log_2 m$ bits. Observe also that, whenever no discrepancy is present at the value n itself, the restricted algorithm automatically provides the right answer for all values $n' \leq n$.

The proof of this theorem results from tail properties of the multinomial distributions and of maxima of geometric random variables.

Assume for instance that we wish to count cardinalities till 2^{27} , that is, over a hundred million, with an accuracy of about 4%. By Theorem 1, one should adopt $m = 1024 = 2^{10}$. Then, each bucket is visited roughly $n/m = 2^{17}$ times. One has $\log_2 \log_2 2^{17} \doteq 4.09$. Adopt $\omega = 0.91$, so that each register has a size of $\ell = 5$ bits, i.e., a value less than 32. Applying the upperbound of the overall probability failure shows that an ℓ -restriction will have little incidence on the result: the probability of a discrepancy⁴ is lower than 12%. In summary: *The basic LOGLOG counting algorithm makes it possible to estimate cardinalities till 10^8 with a standard error of 4% using 1024 registers of 5 bits each, that is, a table of 640 bytes in total.*

5 Algorithmic Engineering

In this section, we describe a concrete implementation of the LOGLOG algorithm that incorporates the probabilistic principles seen in previous sections. At the same time, we propose an optimization that has several beneficial effects: (i) it increases at no extra cost the accuracy of the results, i.e., it decreases the dispersion of the estimates around the mean value; (ii) it allows for the use of smaller register values, thereby improving the storage utilization of the algorithm and nullifying the effect of length restriction discussed in Section 4.

The fundamental probability distribution is that of the value of the M -register in a bucket that receives ν elements (where $\nu \approx n/m$). This is the

⁴ In addition, a correction factor, calculated according to the principles of Section 3, could easily be built into the algorithm, in order to compensate the small bias induced by restriction

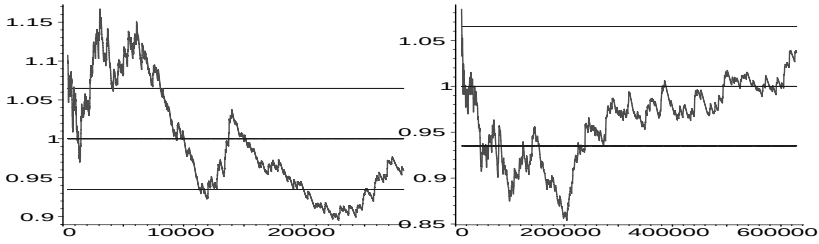


Fig. 2. The evolution of the estimate (divided by the current value of n) provided by super-LOGLOG on all of Shakespeare's works: (left) words; (right) pairs of consecutive words. Here $m = 256$ (standard error=6.5%).

maximum of ν geometric random variables with mean close to $\log_2 n$. The tails of this distribution, though exponential, are still relatively “soft”, as there holds $\mathbb{P}_\nu(M > \log_2 \nu + k) \approx 2^{-k}$. Since the estimate returned involves an exponential of the arithmetic mean of bucket registers, a few exceptional values may still distort the estimate produced by the algorithm, while more tame data will not induce this effect. Altogether, this phenomenon lies at the origin of a natural dispersion of estimates produced by the algorithm, hence it places a limit on the accuracy of cardinality estimates. A simple remedy to the situation consists in using *truncation*:

Truncation Rule. When collecting register values in order to produce the final estimate, retain only the $m_0 := \lfloor \theta_0 m \rfloor$ smallest values and discard the rest. There θ_0 is a real number between 0 and 1, with $\theta_0 = 0.7$ producing near-optimal results. The mean of these registers is computed and the estimate returned is $m_0 \tilde{\alpha}_m 2^{\frac{1}{m_0} \Sigma^* M^{(j)}}$, where Σ^* indicates the truncated sum. The modified constant $\tilde{\alpha}_m$ ensures that the algorithm remains unbiased.

When the truncation rule is applied, accuracy does increase. An empirically determined formula for the standard error is $\frac{1.05}{\sqrt{m}}$, when the Truncation Rule with $\theta_0 = 0.7$ is employed.

Empirical studies justify the fact that register values may be ceiled at the value $\lceil \log_2 \left(\frac{n}{m} \right) \rceil + \delta$, without detectable effect for $\delta = 3$. In other words, one may freely combine the algorithm with restriction as follows:

Restriction Rule. Use register values that are in the interval $[0..B]$, where $\lceil \log_2 \left(\frac{N_{\max}}{m} \right) + 3 \rceil \leq B$.

For instance for the data at the end of Section 4, with $n = 2^{27}$, $m = 1024$, the value $B = 20$ (encoded on 5 bits) is sufficient. But now, the probability that length-restriction affects the estimate of the algorithm drops tremendously.

Fact 1. Combining the basic LOGLOG counting algorithm, the Truncation Rule and the Restriction Rule yields the super-LOGLOG algorithm that estimates cardinalities with a standard error of $\approx \frac{1.05}{\sqrt{m}}$ when m “small bytes” are used. Here a small byte has size $\lceil \log_2 \lceil \log_2 \left(\frac{N_{\max}}{m} \right) + 3 \rceil \rceil$, that is, 5 bits for maximum cardinalities $N_{\max}$ well over 10^8 .

Length of the hash function and collisions. The length H of the hash function—how many bits should it produce?— is guided by previous considerations. There must be $\log_2 m$ bits reserved for bucketing and the bound on register values should be at least as large as the quantity B above. Accordingly this value H must satisfy: $H \geq H_0$, where $H_0 := \log_2 m + \lceil \log_2 \left(\frac{N_{\max}}{m} \right) + 3 \rceil$. In case a value too close to H_0 is adopted (say $0 \leq H - H_0 \leq 3$), then the effect of hashing collisions must be compensated for. This is achieved by inverting the function that gives the expected value of the number of collisions in a hash table (see [3,16] for an analogous discussion). The estimator is then to be changed into $-2^H \log \left(1 - \frac{\tilde{\alpha}_m m}{2^H} 2^{\frac{1}{m}} \Sigma^* M^{(j)} \right)$. (No detectable degradation of performance results from the last modification of the estimator function, and it can safely be used in all cases.)

Risk analysis. For the pure LOGLOG algorithm, the estimate is an empirical mean of random variables that are approximately identically distributed (up to statistical fluctuations in bucket sizes). From there, it can be proved that the quantity $\frac{1}{m} \sum_j M^{(j)}$ is numerically closely approximated by a Gaussian. Consequently, the estimate returned is very roughly Gaussian: at any rate, it has exponentially decaying tails. (In principle, a full analysis would be feasible.) A similar property is expected for the super-LOGLOG algorithm since it is based on the same principles. As a consequence, we obtain the following pragmatic conclusion:

Fact 2. Let $\sigma := \frac{1.05}{\sqrt{m}}$. The estimate is within σ , 2σ , and 3σ of the exact value of the cardinality n in respectively 65%, 95%, and 99% of the cases.

6 Conclusions

That super-LOGLOG performs quite well in practice is confirmed by the following data from simulations:

$k = \log_2 m$	4	5	6	7	8	9	10	11	12
σ^*	29.5	19.8	13.8	9.4	6.5	4.5	3.1	2.2	1.5
$1.05/\sqrt{m}$	26.3	18.6	13.1	9.3	6.5	4.6	3.3	2.3	1.6
Random	22	16	11	8	6	4	3	2.3	2
KingLear	8.2	1.6	2.1	3.9	2.9	1.2	0.3	1.7	—
ShAll	2.9	13.9	4.4	0.9	9.4	4.1	3.0	0.8	0.6
Pi	67	28	9.7	8.6	2.8	5.1	1.9	1.2	0.7

Note. σ^* refers to standard error as estimated from extensive simulations, to be compared to the empirical formula $1.05/\sqrt{m}$. The next lines display the *absolute value* of the relative error measured. **Random** refers to averages over 10,000 runs with $n = 20,000$; the other data are *single* runs: Pi is formed of $2 \cdot 10^7$ records that are consecutive 10-digit slices of the first 200 million decimals of π ; **ShAll** is the whole of Shakespeare’s works. **KingLear** is what its name says. (Naturally, inherent stochastic fluctuations prevent the estimates from always depending

monotonically on memory size (m) in the case of single runs on a given piece of data.)

As we have strived to demonstrate, the LOGLOG algorithm in its optimized version performs quite well. The following table (grossly) summarizes the accuracy (measured by standard error σ) in relation to the storage used for the major methods known. Note that different algorithms operate with different memory units.

<i>Algorithm</i>	<i>Std. Err. (σ)</i>	<i>Memory units</i>	$n = 10^8, \sigma = 0.02$
Adaptive Sampling	$1.20/\sqrt{m}$	Records (≥ 24 -bit words)	10.8 kbytes
Prob. Counting	$0.78/\sqrt{m}$	Words (24–32 bits)	6.0 kbytes
Multires. Bitmap	$\approx 4.4/\sqrt{m}$	Bits	4.8 kbytes
LOGLOG	$1.30/\sqrt{m}$	“Small bytes” (5 bits)	2.1 kbytes
Super-LOGLOG	$1.05/\sqrt{m}$	“Small bytes” (5 bits)	1.7 kbytes

The last column is a rough indication of the storage requirement for an accuracy of 2% and a file of cardinality 10^8 . (The formula for Multiresolution Bitmap is a crude extrapolation based on data of [3].)

Distributing or parallelizing the algorithm is trivial: it suffices to have different processors (sharing the same hash function) operate on different slices of the data and then “max-merge” their tables of registers. Optimal speed-up is clearly attained and interprocess communication is limited to just a few kilobytes. Requirements for an embedded hardware design are absolutely minimal as only addressing, register comparisons, and integer addition are needed.

Acknowledgements. This work has been partly supported by the European Union under the Future and Emerging Technologies programme of the Fifth Framework, ALCOM-FT Project IST-1999-14186. The authors are grateful to Cristian Estan and George Varghese for very liberally sharing ideas and preliminary versions of their works, and to Keith Briggs for his suggestions regarding implementation.

References

1. ALON, N., MATIAS, Y., AND SZEGEDY, M. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences* 58 (1999), 137–147.
2. ESTAN, C., AND VARGHESE, G. New directions in traffic measurement and accounting. In *Proceedings of SIGCOMM 2002* (2002), ACM Press. (Also: UCSD technical report CS2002-0699, February, 2002; available electronically.).
3. ESTAN, C., VARGHESE, G., AND FISK, M. Bitmap algorithms for counting active flows on high speed links. Technical Report CS2003-0738, UCSD, Mar. 2003.
4. FLAJOLET, P. Approximate counting: A detailed analysis. *BIT* 25 (1985), 113–134.
5. FLAJOLET, P. On adaptive sampling. *Computing* 34 (1990), 391–400.
6. FLAJOLET, P., GOURDON, X., AND DUMAS, P. Mellin transforms and asymptotics: Harmonic sums. *Theoretical Computer Science* 144, 1-2 (1995), 3–58.

7. FLAJOLET, P., AND MARTIN, G. N. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences* 31, 2 (1985), 182–209.
8. GIBBONS, P. B., POOSALA, V., ACHARYA, S., BARTAL, Y., MATIAS, Y., MUTHUKRISHNAN, S., RAMASWAMY, S., AND SUEL, T. AQUA: System and techniques for approximate query answering. Tech. report, Bell Laboratories, Murray Hill, New Jersey, Feb. 1998.
9. JACQUET, P., AND SZPANKOWSKI, W. Analytical depoissonization and its applications. *Theoretical Computer Science* 201, 1–2 (1998).
10. KNUTH, D. E. *The Art of Computer Programming*, 2nd ed., vol. 3: Sorting and Searching. Addison-Wesley, 1998.
11. MORRIS, R. Counting large numbers of events in small registers. *Communications of the ACM* 21 (1978), 840–842.
12. MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, 1995.
13. C. R. Palmer, G. Siganos, M. Faloutsos, C. Faloutsos, and P. Gibbons. The connectivity and fault-tolerance of the Internet topology. In *Workshop on Network-Related Data Management* (NRDM-2001).
14. PRODINGER, H. Combinatorics of geometrically distributed random variables: Left-to-right maxima. *Discrete Mathematics* 153 (1996), 253–270.
15. SZPANKOWSKI, W. *Average-Case Analysis of Algorithms on Sequences*. John Wiley, New York, 2001.
16. WHANG, K.-Y., ZANDEN, B. T. V., AND TAYLOR, H. M. A linear-time probabilistic counting algorithm for database applications. *TODS* 15, 2 (1990), 208–229.

Packing a Trunk^{*}

Friedrich Eisenbrand¹, Stefan Funke¹, Joachim Reichel¹, and Elmar Schömer²

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany
`{eisen,funke,reichel}@mpi-sb.mpg.de`

² Universität Mainz, Department of Computer Science, Germany
`schoemer@informatik.uni-mainz.de`

Abstract. We report on a project with a German car manufacturer. The task is to compute (approximate) solutions to a specific large-scale packing problem. Given a polyhedral model of a car trunk, the aim is to pack as many identical boxes of size $4 \times 2 \times 1$ units as possible into the interior of the trunk. This measure is important for car manufacturers, because it is a standard in the European Union.

First, we prove that a natural formal variant of this problem is NP-complete. Further, we use a combination of integer linear programming techniques and heuristics that exploit the geometric structure to attack this problem. Our experiments show that for all considered instances, we can get very close to the optimal solution in reasonable time.

1 Introduction

Geometric packing problems are fundamental tasks in the field of Computational Geometry and Discrete Optimization. The problem we are considering in this paper is of the following type:

Problem 1. Given a polyhedral domain $P \subseteq \mathbb{R}^3$, which is homeomorphic to a ball, place as many boxes of size $4 \times 2 \times 1$ into P such that no two of them intersect.

We were approached with this problem by a car manufacturer whose problem was to measure the volume of a trunk according to a European standard (DIN 70020). The intention of this standard is that the continuous volume of a trunk does not reflect the actual storage capacity, since the baggage, which has to be stored, is usually discrete. The European standard asks for the number of $200\text{mm} \times 100\text{mm} \times 50\text{mm} = 1$ liter boxes, which can be packed into the trunk. Up till now, this problem is solved manually with a lot of effort.

Contributions

We show that Problem 1 is NP-complete by a reduction to 3-SAT. Further, we attack this problem on the basis of an integer linear programming formulation.

^{*} This work was partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

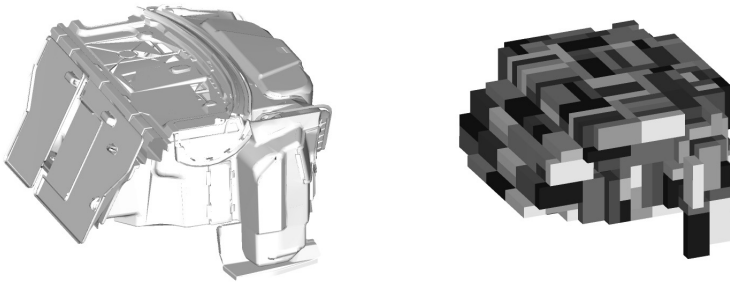


Fig. 1. CAD model of a trunk and a possible packing of boxes

It turns out that the pure ILP approach does not work, even with the use of problem-specific cutting planes in a branch-and-cut framework. We therefore design and evaluate several heuristics based on the LP-relaxation and the geometric structure of the problem. The combination of the exact ILP-approach and the herein proposed heuristics yield nearly optimal solutions in reasonable time.

Related Work

Various versions of packing problems have been shown to be NP-complete [1]. We derive our complexity result from an NP-complete packing variant inspected by FOWLER, PATERSON and TANIMOTO [2]. Here the task is to pack unit squares in the plane. In their work, they do not consider rectangular objects that are not squares and the region which has to be packed is not homeomorphic to a disk.

Other theoretical and practical results in the area of industrial packing problems suggest that allowing arbitrary placements and orientations of the objects even within a two-dimensional domain is only viable for extremely small problem instances. For example, DANIELS and MILENKOVIC consider in [3,4] the problem of minimizing cloth utilization when cutting out a small number of pieces from a roll of stock material. If arbitrary placements and orientations are allowed only very small problem instances (≤ 10 objects) can be handled. For larger problem instances they discretize the space of possible placements and use heuristics to obtain solutions for up to 100 objects.

A survey of the application of generic optimization techniques like simulated annealing, genetic algorithms, gradient methods, etc. to our type of packing problems can be found in [5]. AARDAL and VERWEIJ consider in [6] the problem of labelling points on a map with pairwise disjoint rectangles such that the corners of the rectangles always touch the point they are labelling. Similar to our work, this discretization of the possible placements reduces the problem to finding a maximum stable set in the intersection/conflict graph (which for this application is much smaller than in our case, though).

2 Continuous Box Packing Is NP-Complete

In this section, we prove that Problem 1 is NP-complete. In a first step, we show that a two-dimensional discrete variant of this problem is NP-complete.

Definition 1 ($m \times n$ -Rectangle-Packing). *Given integers $k, m, n \in \mathbb{N}, m \geq n$ and sets $H \subseteq \mathbb{Z}^2, V \subseteq \mathbb{Z}^2$, decide whether it is possible to pack at least k axis-aligned boxes of size $m \times n$ in such a way that the lower left corner of a horizontal (vertical) box coincides with a point in H respectively V .*

This problem is trivial for $m = n = 1$. For $m = 2, n = 1$, the problem can be formulated as set packing problem with sets of size 2 and solved in polynomial time by matching techniques [1].

Proposition 1. 3×3 -RECTANGLE-PACKING and 8×4 -RECTANGLE-PACKING are NP-complete.

Proof. The case $m = n = 3$ has been shown by FOWLER et al. [2] using a reduction of 3-SAT to this problem. We use the same technique here for $m = 8, n = 4$ and refer to their article for the details.

Given a formula in conjunctive normal form (CNF) with three literals per clause, FOWLER et al. construct a planar graph as follows. For each variable x_n there is a cycle of even length. Such a cycle has two stable sets of maximal cardinality which correspond to the two possible assignments of x_n . Moreover, paths of two cycles can cross each other at so called *crossover regions*, which have the special property that for each maximum stable set both paths do not influence each other. For each clause exists a *clause region*, that increases the value of a maximum stable set iff the clause is satisfied. The number k can be easily deduced from the number of nodes, crossover and clause regions. The proof is completed by explaining how to compute the sets H and V such that the constructed graph is the intersection graph of the packing problem. Thus the formula is satisfiable iff there exists a packing of cardinality $\geq k$.

For the case $m = 8, n = 4$, we need to explain how to construct the crossover and clause region. The crossover region for two cycle paths are realized as shown in Fig. 2. Note that the rectangle in the center has size 9×5 and allows four different placements of a 8×4 rectangle. Clause regions are constructed as shown in Fig. 3. Both constructions maintain their special properties as in the case of 3×3 squares and the remainder of the proof is identical. $\square$

The decision variant of Problem 1 is defined as follows:

Definition 2 (Continuous-Box-Packing). *Given a polyhedral domain $P \subseteq \mathbb{R}^3$, which is homeomorphic to a ball, and $k \in \mathbb{N}$, decide whether it is possible to pack at least k boxes of size $4 \times 2 \times 1$ into P such that no two of them intersect.*

Theorem 1. CONTINUOUS-BOX-PACKING is NP-complete.

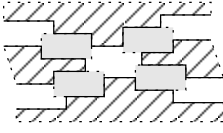


Fig. 2. Crossover region and corresponding intersection graph

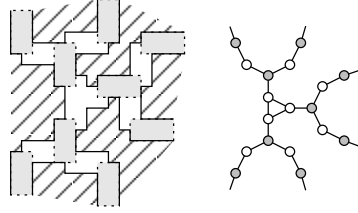


Fig. 3. Clause region and corresponding intersection graph

Proof. We reduce the problem to 8×4 -RECTANGLE-PACKING. Let (k, H, V) denote an instance of 8×4 -RECTANGLE-PACKING. Intuitively our approach works as follows: We extrude the shape induced by the sets H and V into the third dimension with z -coordinates ranging from 0 to 1. On the bottom of this construction we glue a box of height $\frac{1}{2}$. The size of the box is chosen such that the construction is homeomorphic to a ball. More formally, $P \subseteq \mathbb{R}^3$ is constructed as follows:

$$P_H := \bigcup_{(x,y) \in H} \left[\frac{1}{2}x, \frac{1}{2}x + 4 \right] \times \left[\frac{1}{2}y, \frac{1}{2}y + 2 \right] \times [0, 1] ,$$

$$P_V := \bigcup_{(x,y) \in V} \left[\frac{1}{2}x, \frac{1}{2}x + 2 \right] \times \left[\frac{1}{2}y, \frac{1}{2}y + 4 \right] \times [0, 1] ,$$

$$P := P_H \cup P_V \cup X \times Y \times \left[-\frac{1}{2}, 0 \right] ,$$

where X and Y denote the projection of $P_H \cup P_V$ onto the first respectively second coordinate. This construction can be carried out in polynomial time. It is clear that a projection of a maximal packing to the first two coordinates corresponds to a maximal packing of the two-dimensional problem of the same value and vice versa. $\square$

3 From the CAD Model to the Maximum Stable Set Problem

The data we obtain from our industry partner is a CAD model of the trunk to be packed. Since the manual packings used so far had almost all boxes axis aligned to some coordinate system, we decided to discretize the problem in the following way. We first **Discretize the Space** using a three-dimensional cubic grid. Given the box extensions of 200 mm, 100 mm and 50 mm, a grid width of 50 mm was an obvious choice. In order to improve the approximation of the trunk by this grid, we also work with a refined grid of edge length 25 mm. Even smaller grid widths did not improve the results substantially and for larger CAD models the number of cubes became too big. In the following, numbers depending on the

grid granularity refer to a grid of edge length 50 mm and numbers for the refined grid of edge length 25 mm are added in parentheses.

The alignment of the coordinate axes is done such that the number of cubes which are completely contained in the interior of the trunk model is maximized. In practice, the best cubic grids were always aligned with the largest almost planar boundary patch of the trunk model – which most of the time was the bottom of the trunk. For the remaining translational and one-dimensional rotational freedom we use an iterative discrete procedure to find the best placement of the grid. The result of this phase is an approximation of the trunk interior as depicted in Fig. 4.

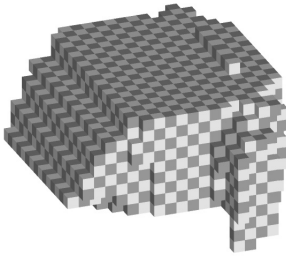


Fig. 4. Interior approximation of the trunk of Fig. 1

In the next phase we use the cubic grid to **Discretize the Box Placements**. A box of dimension 200mm $\times$ 100mm $\times$ 50mm can be viewed as a box consisting of $4 \times 2 \times 1$ ($8 \times 4 \times 2$) cubes of the cubic grid. We will only allow placements of boxes such that they are aligned with the cubic grid. So the placement of one box is defined by six parameters (x, y, z, w, h, d) , where (x, y, z) denotes the position of a cube in our grid – we will call this the *anchor* of the box – and (w, h, d) denotes how far the box extends to the right, to the top and in depth. (w, h, d) can be any permutation of $\{4, 2, 1\}$ ($\{8, 4, 2\}$), so for a given anchor, there are 6 possible orientations of

how to place a box at that position.

Our goal is now to place as many such boxes in the manner described above in our cubic grid such that each box consists only of cubes that approximate the interior of the trunk and no pair of boxes shares a cube.

It is straightforward to formalize this problem using the following construction: The *conflict graph* $G(\mathcal{G}) = (V, E)$ for a cubic grid $\mathcal{G}$ is constructed as follows. There is a node $v_{x,y,z,w,h,d} \in V$ iff the box placed at anchor (x, y, z) with extensions (w, h, d) consists only of cubes located inside the trunk. Two nodes v and w are adjacent iff the boxes associated with v and w intersect.

A *stable* or *independent* set $S \subseteq V$ of a graph $G = (V, E)$ is a subset of the nodes of G which are pairwise nonadjacent. The *stable set problem* is the problem of finding a stable set of a graph G with maximum cardinality. It is NP-hard [1].

There is a one-to-one relationship between the stable sets in $G(\mathcal{G})$ and valid box packings in $\mathcal{G}$, in particular every stable set in $G(\mathcal{G})$ has a corresponding valid box packing in $\mathcal{G}$ of same size and vice versa. We use this one-to-one relationship to reduce the maximum box packing problem to a maximum stable set problem on a graph:

Lemma 1. *The maximum box packing problem for a grid $\mathcal{G}$ can be reduced to a maximum stable set problem in the corresponding conflict graph $G(\mathcal{G})$.*

To give an idea about the sizes of the conflict graphs we are dealing with, we show in Table 1 the sizes of the conflict graphs for our (rather small) trunk model M1 and grid widths 50 mm and 25 mm. We will use the 50 mm discretization of this model as a running example throughout the presentation of all our algorithms in the next section.

Table 1. Grid and Conflict Graph sizes for trunk model M1

grid granularity [mm]	# interior cubes	# nodes in $G(\mathcal{G})$	# edges in $G(\mathcal{G})$
50	2210	8787	649007
25	19651	68548	62736126

4 Solving the Stable-Set Problem

4.1 A Branch-and-Cut Algorithm

In the previous section, we modeled our packing problem as a maximum stable-set problem for the conflict-graph $G(\mathcal{G}) = (V, E)$ for a given grid $\mathcal{G}$. In this Section we describe how we attack this stable-set problem with a branch-and-cut algorithm.

The stable-set problem has the following well known integer programming formulation, see, e.g. [7]:

$$\begin{aligned}
 & \max \sum_{v \in V} x_v \\
 & \{u, v\} \in E : x_u + x_v \leq 1 \\
 & u \in V : x_u \in \{0, 1\} .
 \end{aligned} \tag{1}$$

It is easy to see that the characteristic vectors of stable sets of G are exactly the solutions to this constraint system.

Standard ILP solvers try to solve this problem using techniques like branch-and-bound. These techniques depend heavily on the quality of the *LP relaxation*. Therefore, it is beneficial to have a relaxation which is strong. We pursue this idea via incorporating clique inequalities and (lifted) odd-hole inequalities.

Clique inequalities. A *clique* C of G is a subset of the nodes $C \subseteq V$, such that every two nodes in C are connected. If S is a stable set and C is a clique, then there can be at most one element of S which also belongs to C . This observation implies the constraints

$$\sum_{v \in C} x_v \leq 1 \text{ for each } C \in \mathcal{C} , \tag{2}$$

where $\mathcal{C}$ is the set of cliques of G .

If C is a maximal clique, then the corresponding clique inequality (2) defines a facet of the convex hull of the characteristic vectors χ^S of stable sets S of G , see [8]. Thus the clique inequalities are strong in the sense that they cannot be implied by other valid inequalities. The number of maximal cliques can be exponential and furthermore, the *separation problem* for the clique inequalities is NP-hard for general graphs [9]. However, in our application the number of cliques is polynomial and the maximum cliques can be enumerated in polynomial time. This result is established with the following lemma. A proof is straightforward.

Lemma 2. *Every maximal clique in $G(\mathcal{G})$ corresponds to the box placements in $\mathcal{G}$ which overlap one particular cube.*

Therefore we can strengthen the formulation (1) by replacing the edge constraints with the clique constraints (2) and obtain the polynomial *clique formulation*.

Odd hole inequalities. An *odd hole* [10] H of G is a cordless cycle of G with an odd number of nodes. If S is a stable set of G , then there can be at most $\lfloor |H|/2 \rfloor$ elements of S belonging to H . This implies the constraints

$$\sum_{v \in H} x_v \leq \lfloor |H|/2 \rfloor \text{ for all } H \in \mathcal{H}, \quad (3)$$

where $\mathcal{H}$ denotes the set of odd holes of G . These inequalities can be strengthened with a *sequential lifting* process, suggested in [8,11], see also [12]. We apply the algorithm of GERARDS and SCHRIJVER [13] to identify nearly violated odd hole inequalities and strengthen them using different lifting sequences.

For our running example M1 / 50mm, the LP relaxation yields an upper bound of 268 liters. Running our branch-and-cut approach for 24 hours and taking the best result found, we obtained a packing of 266 liters.

4.2 Heuristics

Unfortunately, the above described branch-and-cut algorithm works only for small packing instances and not for trunks of real-world size. On the other hand, the ILP-approach is an exact algorithm for the discretized problem that we wish to solve. In the following we propose several heuristics for our problem that can be combined with our exact ILP-approach. Depending on the employed heuristic we obtain trade-offs between running time and solution quality.

Partitioned ILP Formulations

This approach partitions the packing problem into independent sub-problems, which are exactly solved with branch-and-cut individually and thereafter combined. We partition a given grid by axis-parallel planes into smaller sections. Tests have shown that one should choose the sizes of the sections ranging from 50 to 100 liters.

But the cutting of the grid into smaller sections leads to waste and obtained solutions can be further improved by local optimization across section boundaries. By moving some of the packed boxes it is possible to bring several uncovered cubes into proximity, such that one more box can be packed. This inspired the following modification.

We slightly change the objective function of (1) such that some packings of a given cardinality are preferred. The old objective function is replaced by $\sum_{v \in V} c_v x_v$, i.e. we solve a *weighted* stable set problem and the coefficients $c_v \in \mathbb{R}$ are computed as follows. Assume the box corresponding to node v is anchored at (x, y, z) and contained in the section $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [z_{\min}, z_{\max}]$. The coefficient c_v is then computed as

$$c_v = 1 + \frac{1}{u} \cdot \frac{x_{\max} - x + y_{\max} - y + z_{\max} - z}{x_{\max} - x_{\min} + y_{\max} - y_{\min} + z_{\max} - z_{\min}}, \quad (4)$$

where u is an upper bound for the optimal value of (1) for this section. The new objective function still aims for packings with largest cardinality, but among packings of the same cardinality those with boxes anchored as near as possible to $(x_{\min}, y_{\min}, z_{\min})$ are preferred. Thus uncovered cubes tend to appear near the $x = x_{\max}$, $y = y_{\max}$ and $z = z_{\max}$ boundaries of the section and can be reused when processing the adjacent sections.

Using this approach, we achieved a packing of 267 boxes with 24 hours run-time. Although the quality of the solution has been improved by a small amount, it takes too long to achieve good results.

A Greedy Algorithm

The most obvious idea for a heuristic for the stable set problem in a graph is to use a greedy approach. The greedy algorithm selects a vertex with smallest degree and adds it to the stable set S determined so far, then removes this vertex and all its neighbors and repeats.

The algorithm tends to place boxes first close to the boundary and then growing to the inside until the trunk is filled. This is due to the fact that placements close to the boundary 'prohibit' fewer other placements and therefore their degree in the conflict graph is rather low.

There is a bit of ambiguity in the formulation of the Greedy algorithm. If there are several vertices with minimum degree, we can choose the next vertex uniformly at random. This randomized version of the greedy algorithm is repeated several times.

As one might expect, the Greedy algorithm is very fast, but outputs a result of rather low quality. We achieved a solution of 259 liters in less than a minute. The maximum of ten runs of the randomized version was 262 liters.

Geometry-Guided First-Level Heuristics

Looking at the model of the trunk, one observes that it is quite easy to tightly pack some boxes in the center of the trunk, whereas difficulties arise when ap-

proaching the irregular shape of the boundary. The following two heuristics exploit this fact by filling the center of the trunk with a solid block of boxes. This approach significantly decreases the problem complexity.

Easyfill. This algorithm strongly simplifies the problem by restricting the set of allowed placements for boxes. Supposing the first box is packed at (x, y, z, w, h, d) , we solely consider boxes with the same orientation anchored at $(x + \mathbb{Z}w, y + \mathbb{Z}h, z + \mathbb{Z}d)$. This leads to a tight packing in the interior of the trunk and the remaining cubes near to the boundary can be packed by one of the other algorithms.

For each of the 6 orientations, there are 8 (64) possibilities to align the first box on the grid. The quality of the results heavily depends on the placement of the first box. Thus we repeat the procedure with all different placements of the first box.

As it turns out, if one exactly uses this algorithm, the remaining space is not sufficient to place many additional boxes, but a lot of cubes are left uncovered. To overcome this problem, we use the following approach. In the first phase we peel off some layers of the cubes representing the interior of the trunk, and then run the Easyfill algorithm. In the second phase we re-attach the peeled-off layers again and fill the remaining part using some other algorithm.

By using Easyfill we were able to improve the results obtained so far. In combination with the Greedy algorithm, we achieved a solution of 263 liters in 1 minute. A better solution of 267 boxes was achieved in combination with the ILP algorithm. Here we also had to terminate the branch-and-cut phase after about 30 minutes for each combination of orientation and alignment of the first box to limit the total running time to 24 hours.

Matching. Another interesting idea to get a compact packing of a large part of the trunk is to cluster two boxes to a larger box consisting of $4 \times 2 \times 2$ ($8 \times 4 \times 4$) cubes and then interpret these boxes as $2 \times 1 \times 1$ cubes on a coarser grid of side length 100 mm (50 mm). As we have seen in Section 2, this special packing problem can be solved in polynomial time using a maximum cardinality matching.

Similar to the Easyfill algorithm, there are 8 (64) possibilities to align the coarse grid with the original grid. Likewise, there is little freedom for packing the remaining cubes and we use the same approach as in the case of the Easyfill algorithm.

The results for the Matching approach combined with Greedy algorithms are comparable to the Easyfill approach. So we obtained a volume of 263 liters with a slightly better running time. In combination with the ILP approach we get a slightly worse result of 265 liters.

LP Rounding

As solving the ILP to optimality is pretty hard, one might wonder how to make use of an optimal solution to the LP relaxation – which can be obtained in

reasonable time – to design a heuristic. One way is to solve the LP and round the possibly fractional values of the optimal solution to 0/1-values, of course obeying the stable set constraints. This heuristic is implemented in the ILP-solver, but is not very effective. So we came up with the following iterative procedure:

1. solve the clique LP for G to optimality
2. let B be the box placements corresponding to the 5 % largest LP values
3. use greedily as many placements from B as possible, remove their corresponding vertices and neighbors from G and goto 1

This approach took 45 minutes to compute a solution of 268 boxes. This is the value of the LP relaxation and thus optimal.

5 Experimental Evaluation

In this section, we present experimental results showing the performance of the algorithms. We present results for three models, named M1, M2 and M3 in the following, with grids of granularity of 50 mm and 25 mm.

Table 2 shows some characteristics of both models. Model M2 is about 40% larger than model M1 and model M3 about three times larger than M2. Note that refining the grid granularity quickly increases the size of the conflict graph and enlarges the grid volume, whereas the upper bound obtained by the LP relaxation does not grow by the same factor.

Table 2. Some characteristics of the used models

model	M1	M1	M2	M2	M3
grid granularity [mm]	50	25	50	25	50
# nodes in $G(\mathcal{G})$	8787	68548	12857	95380	44183
# edges in $G(\mathcal{G})$	649007	62736126	974037	88697449	3687394
grid volume [l]	276	307	396	429	1214
upper bound (LP relaxation) [l]	268	281	389	398	1202
best solution [l]	268	271	384	379	1184

For model M1 our industrial partner provided a manually achieved solution of 272 liters (which applies to the original trunk model only, not the discretized grid), whereas our best solution has a value of 271 liters.

In Table 3 we present results for the GREEDY, RANDOMIZED GREEDY, LP ROUNDING and ILP algorithm – standalone as well as combined with the MATCHING and EASYFILL algorithm. The table is completed by the data for the PARTITIONED ILP algorithm. Each run was stopped after at last 24 hours and in this case, the so far best result is reported.

All algorithms using LP- or ILP-based techniques were run on a SunFire 15000 with 900 MHz SPARC III+ CPUs, using the operating system SunOS

Table 3. Computed trunk volumes (in liters) and running-times (in minutes). For the results marked with an asterisk (*), we stopped the computation after 24 hours and took the best result found so far.

model	M1				M2				M3	
grid granularity [mm]	50		25		50		25		50	
	vol.	time	vol.	time	vol.	time	vol.	time	vol.	time
GREEDY	259	1	262	2	373	1	364	2	1148	1
EASYFILL + GREEDY	263	1	269	61	378	1	376	79	1169	1
MATCHING + GREEDY	263	1	268	5	375	1	373	7	1167	1
RANDOMIZED GREEDY	262	1	265	19	377	1	365	27	1147	2
EASYFILL + RAND. GR.	264	5	271	807	381	5	377	1038	1171	26
MATCHING + RAND. GR.	263	1	268	67	377	1	373	83	1165	4
LP ROUNDING	268	45	–	–	384	427	–	–	1184	189
EASYFILL + LP ROUND.	267	29	269	24h*	384	48	376	24h*	1184	95
MATCHING + LP ROUND.	265	35	267	453	383	2	379	792	1178	8
ILP	266	24h*	–	–	384	24h*	–	–	1136	24h*
EASYFILL + ILP	267	24h*	269	24h*	383	24h*	379	24h*	1180	24h*
MATCHING + ILP	265	24h*	270	24h*	383	24h*	379	24h*	1176	24h*
PARTITIONED ILP	267	24h*	260	24h*	384	24h*	378	24h*	1175	24h*

5.9. CPLEX 8.0 was used as (I)LP-solver. All other algorithms were run on a Dual Xeon 1.7 GHz under Linux 2.4.18. Our implementation is single-threaded and thus does not make use of multiple CPUs.

For the MATCHING and EASYFILL algorithms, one layer of cubes was peeled off for the first phase. This has turned out as a good compromise between enough freedom and not too large complexity for the second phase. For RANDOMIZED GREEDY, the best results of ten runs are reported.

One observes that the randomization of the GREEDY algorithm leads to better results while the runtime increases according to the number of runs. Both algorithms can be improved by applying EASYFILL or MATCHING first. This imposes a further increase in the running time, due to the many subproblems that have to be solved.

However, all GREEDY algorithms are outperformed by (I)LP-based techniques, whereas LP ROUNDING is significantly faster than the ILP algorithm. Combining both algorithms with EASYFILL and MATCHING leads to worse results on a grid with granularity 50 mm, whereas it is absolutely necessary on the refined grid due to its huge complexity. The results obtained by PARTITIONED ILP are comparable to LP ROUNDING and ILP.

6 Conclusion

In this paper we have considered the problem of maximizing the number of boxes of a certain size that can be packed into a car trunk. We have shown that

this problem is NP-complete. Our first approach which was based on an ILP formulation of the problem did not turn out to be very practical. Therefore we have designed several heuristics based on the ILP formulation and the geometric structure of the problem. In this way we obtained good trade-offs between running time and quality of the produced solution. In fact, at the end we could compute solutions as good or even better than the best ILP based solutions within a certain time frame.

There are still a number of interesting problems left open. In our problem instances, we could restrict to only axis-aligned placements of the boxes without sacrificing too much of the possible volume, but there might be other instances (maybe not of trunk-type) where this is a too severe restriction.

References

1. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness. Freeman (1979)
2. Fowler, R.F., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. *Information Processing Letters* **12** (1981) 133–137
3. Milenkovic, V.J.: Rotational polygon containment and minimum enclosure using only robust 2d constructions. *Computational Geometry* **13** (1999) 3–19
4. Daniels, K., Milenkovic, V.J.: Column-based strip packing using ordered and compliant containment. In: 1st ACM Workshop on Applied Computational Geometry (WACG). (1996) 33–38
5. Cagan, J., Shimada, K., Yin, S.: A survey of computational approaches to three-dimensional layout problems. *Computer-Aided Design* **34** (2002) 597–611
6. Verweij, B., Aardal, K.: An optimisation algorithm for maximum independent set with applications in map labelling. In: *Algorithms—ESA '99* (Prague). Volume 1643 of *Lecture Notes in Comput. Sci.* Springer, Berlin (1999) 426–437
7. Grötschel, M., Lovász, L., Schrijver, A.: *Geometric Algorithms and Combinatorial Optimization*. Volume 2 of *Algorithms and Combinatorics*. Springer (1988)
8. Padberg, M.W.: On the facial structure of set packing polyhedra. *Mathematical Programming* **5** (1973) 199–215
9. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* **1** (1981) 169–197
10. Chvátal, V.: On certain polytopes associated with graphs. *Journal of Combinatorial Theory Ser. B* **18** (1975) 138–154
11. Wolsey, L.: Faces for a linear inequality in 0-1 variables. *Mathematical Programming* **8** (1975) 165–178
12. Nemhauser, G.L., Wolsey, L.A.: *Integer programming*. In et al., G.L.N., ed.: *Optimization*. Volume 1 of *Handbooks in Operations Research and Management Science*. Elsevier (1989) 447–527
13. Gerards, A.M.H., Schrijver, A.: Matrices with the Edmonds-Johnson property. *Combinatorica* **6** (1986) 365–379

Fast Smallest-Enclosing-Ball Computation in High Dimensions^{*}

Kaspar Fischer^{1**}, Bernd Gärtner¹, and Martin Kutz^{2***}

¹ ETH Zürich, Switzerland

² FU Berlin, Germany

Abstract. We develop a simple combinatorial algorithm for computing the smallest enclosing ball of a set of points in high dimensional Euclidean space. The resulting code is in most cases faster (sometimes significantly) than recent dedicated methods that only deliver approximate results, and it beats off-the-shelf solutions, based e.g. on quadratic programming solvers. The algorithm resembles the simplex algorithm for linear programming; it comes with a Bland-type rule to avoid cycling in presence of degeneracies and it typically requires very few iterations. We provide a fast and robust floating-point implementation whose efficiency is based on a new dynamic data structure for maintaining intermediate solutions.

The code can efficiently handle point sets in dimensions up to 2,000, and it solves instances of dimension 10,000 within hours. In low dimensions, the algorithm can keep up with the fastest computational geometry codes that are available.

1 Introduction

The problem of finding the smallest enclosing ball (SEB, a.k.a. minimum bounding sphere) of a set of points is a well-studied problem with a large number of applications; if the points live in low dimension d ($d \leq 30$, say), methods from computational geometry yield solutions that are quite satisfactory in theory and in practice [1,2,3,4,5]. The case $d = 3$ has important applications in graphics, most notably for visibility culling and bounding sphere hierarchies.

There are a number of very recent applications in connection with *support vector machines* that require the problem to be solved in higher dimensions; these include e.g. high-dimensional clustering [6,7] and nearest neighbor search [8], see also the references in Kumar et al. [9].

^{*} Partly supported by the IST Programme of the EU and the Swiss Federal Office for Education and Science as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG – Effective Computational Geometry for Curves and Surfaces).

^{**} Supported by the Berlin/Zürich joint graduate program “Combinatorics, Geometry, and Computation” (CGC).

^{***} Member of the European graduate school “Combinatorics, Geometry, and Computation” supported by the Deutsche Forschungsgemeinschaft, grant GRK588/2.

The existing computational geometry approaches cannot (and were not designed to) deal with most of these applications because they become inefficient already for moderately high values of d . While codes based on Welzl’s method [3] cannot reasonably handle point sets beyond dimension $d = 30$ [4], the quadratic programming (QP) approach of Gärtner and Schönherr [5] is in practice polynomial in d . However, it critically requires arbitrary-precision linear algebra to avoid robustness issues, which limits the tractable dimensions to $d \leq 300$ [5].

Higher dimensions can be dealt with using state-of-the-art floating point solvers for QP, like e.g. the one of CPLEX [10]. It has also been shown that the SEB problem is an instance of *second order cone programming* (SOCP), for which off-the-shelf solutions are available as well, see [11] and the references there.

It has already been observed by Zhou et al. that general-purpose solvers can be outperformed by taking the special structure of the SEB problem into account. Their result [11] is an interior point code which can even handle values up to $d = 10,000$. The code is designed for the case where the number of points is not larger than the dimension; test runs only cover this case and stop as soon as the ball has been determined up to a fixed accuracy. Zhou et al.’s method also works for computing the approximate smallest enclosing ball of a set of balls.

The recent polynomial-time $(1 + \epsilon)$ -approximation algorithm of Kumar et al. goes in a similar direction: it uses additional structure (in this case *core sets*) on top of the SOCP formulation in order to arrive at an efficient implementation for higher d (test results are only given up to $d = 1,400$) [9].

In this paper, we argue that in order to obtain a fast solution even for very high dimensions, it is not necessary to settle for suboptimal balls: we compute the *exact* smallest enclosing ball using a *combinatorial* algorithm. Unlike the approximate methods, our algorithm constitutes an exact method in the RAM model; and our floating-point implementation shows very stable behaviour. This implementation beats off-the-shelf interior-point-based methods as well as Kumar et. al.’s approximate method; only for $d \geq 1,000$, and if the number of points does not considerably exceed d , our code is outperformed by the method of Zhou et al. (which, however, only computes approximate solutions).

Our algorithm—which is a pivoting scheme resembling the simplex method for linear programming—actually computes the set of at most $d + 1$ *support points* whose circumsphere determines the smallest enclosing ball. The number of iterations is small in practice, but there is no polynomial bound on the worst-case performance.

The idea behind the method is simple and a variant has in fact already been proposed by Hopp et al. as a heuristic, along with an implementation for $d = 3$, but without any attempts to prove correctness and termination [12]: start with a balloon strictly containing all the points and then deflate it until it cannot shrink anymore without losing a point. Our contribution is two-fold. On the theoretical side, we develop a pivot rule which guarantees termination of the method even under degeneracies. In contrast, a naive implementation might cycle (Hopp et al. ignore this issue). The rule is Bland’s rule for the simplex method [13], adapted to our scenario, in which case the finiteness has an appealing geomet-

ric proof. On the practical side, we represent intermediate solutions (which are affinely independent point sets, along with their circumcenters) in a way that allows fast and robust updates under insertion or deletion of a single point. Our representation is an adaptation of the QR -factorization technique [14]. Already for $d = 3$, this makes our code much faster than that of Hopp et al. and we can efficiently handle point sets in dimensions up to $d = 2,000$. Within hours, we are even able to compute the SEB for point sets in dimensions up to 10,000, which is the highest dimension for which Zhou et al. give test results [11].

2 Sketch of the Algorithm

This section provides the basic notions and a central fact about the SEB problem. We briefly sketch the main idea of our algorithm, postponing the details to Section 3.

Basics. We denote by $B(c, r) = \{x \in \mathbb{R}^d \mid \|x - c\| \leq r\}$ the d -dimensional ball of center $c \in \mathbb{R}^d$ and radius $r \in \mathbb{R}_+$. For a point set T and a point c in $\mathbb{R}^d$, we write $B(c, T)$ for the ball $B(c, \max_{p \in T} \|p - c\|)$, i.e., the smallest ball with given center c that encloses the points T . The *smallest enclosing ball* $\text{SEB}(S)$ of a finite point set $S \subset \mathbb{R}^d$ is defined as the ball of minimal radius which contains the points in S , i.e., the ball $B(c, S)$ of smallest radius over all $c \in \mathbb{R}^d$. The existence and uniqueness of $\text{SEB}(S)$ are well-known [3], and so is the following fact which goes back to Seidel [5].

Lemma 1 (Seidel). *Let T be a set of points on the boundary of some ball B with center c . Then $B = \text{SEB}(T)$ if and only if $c \in \text{conv}(T)$.*

We provide a simple observation about convex combinations of points on a sphere, which will play a role in the termination proof of our algorithm.

Lemma 2. *Let T be a set of points on the boundary of some ball with positive radius and with center $c \in \text{conv}(T)$. Fix any set of coefficients such that*

$$c = \sum_{p \in T} \lambda_p p, \quad \sum_{p \in T} \lambda_p = 1, \quad \forall p \in T: \lambda_p \geq 0.$$

Then $\lambda_p \leq 1/2$ for all $p \in T$.

The pivot step. The *circumsphere* $\text{CS}(T)$ of a nonempty affinely independent set T is the unique sphere with center in the affine hull $\text{aff}(T)$ that goes through the points in T ; its center is called the *circumcenter* of T , denoted by $\text{cc}(T)$. A nonempty affinely independent subset T of the set S of given points will be called a *support set*.

Our algorithm steps through a sequence of pairs (T, c) , maintaining the invariant that T is a support set and c is the center of a ball B containing S and having T on its boundary. Lemma 1 tells us that we have found the smallest enclosing ball when $c = \text{cc}(T)$ and $c \in \text{conv}(T)$. Until this criterion is fulfilled, the algorithm performs an iteration (a pivot step) consisting of a *walking phase* which is preceded by a *dropping phase* in case $c \in \text{aff}(T)$.

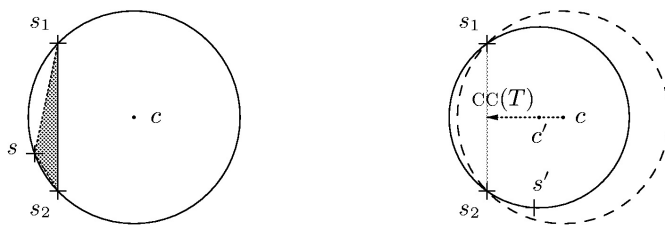


Fig. 1. Dropping s from $T = \{s, s_1, s_2\}$ (left) and walking towards the center $\text{CC}(T)$ of the circumsphere of $T = \{s_1, s_2\}$ until s' stops us (right).

Dropping. If $c \in \text{aff}(T)$, the invariant guarantees that $c = \text{CC}(T)$. Because $c \notin \text{conv}(T)$, there is at least one point $s \in T$ whose coefficient in the affine combination of T forming c is negative. We drop such an s and enter the walking phase with the pair $(T \setminus \{s\}, c)$, see left of Fig. 1.

Walking. If $c \notin \text{aff}(T)$, we move our center on a straight line towards $\text{CC}(T)$. Lemma 3 below establishes that the moving center is always the center of a (progressively smaller) ball with T on its boundary. To maintain the algorithm's invariant, we must stop walking as soon as a new point $s' \in S$ hits the boundary of the shrinking ball. In that case we enter the next iteration with the pair $(T \cup \{s'\}, c')$, where c' is the stopped center; see Fig. 1. If no point stops the walk, the center reaches $\text{aff}(T)$ and we enter the next iteration with $(T, \text{CC}(T))$.

3 The Algorithm in Detail

Let us start with some basic facts about the walking direction from the current center c towards the circumcenter of the boundary points T .

Lemma 3. *Let T be a nonempty affinely independent point set on the boundary of some ball $B(c, r)$, i.e., $T \subset \partial B(c, r) = \partial B(c, T)$. Then*

- (i) *the line segment $[c, \text{CC}(T)]$ is orthogonal to $\text{aff}(T)$,*
- (ii) *$T \subset \partial B(c', T)$ for each $c' \in [c, \text{CC}(T)]$,*
- (iii) *radius($B(\cdot, T)$) is a strictly monotone decreasing function on $[c, \text{CC}(T)]$, with minimum attained at $\text{CC}(T)$.*

Note that part (i) of this lemma implies that the circumcenter of T coincides with the orthogonal projection of c onto $\text{aff}(T)$, a fact that will become important for our actual implementation.

When moving along $[c, \text{CC}(T)]$, we have to check for new points to hit the shrinking boundary. The subsequent lemma tells us that all points “behind” $\text{aff}(T)$ are uncritical in this respect, i.e., they cannot hit the boundary and thus cannot stop the movement of the center. Hence, we may ignore these points during the walking phase.

```

procedure seb( $S$ );
begin
   $c :=$  any point of  $S$ ;
   $T := \{p\}$ , for a point  $p$  of  $S$  at maximal distance from  $c$ ;
  while  $c \notin \text{conv}(T)$  do
    [ Invariant:  $B(c, T) \supset S$ ,  $\partial B(c, T) \supset T$ , and  $T$  affinely independent ]
    if  $c \in \text{aff}(T)$  then drop a point  $q$  from  $T$  with  $\lambda_q < 0$  in (2);
    [ Invariant:  $c \notin \text{aff}(T)$  ]
    among the points in  $S \setminus T$  that do not satisfy (1) find one,  $p$  say,
    that restricts movement of  $c$  towards  $\text{CC}(T)$  most, if one exists;
    move  $c$  as far as possible towards  $\text{CC}(T)$ ;
    if walk has been stopped then  $T := T \cup \{p\}$ ;
  end while;
  return  $B(c, T)$ ;
end seb;

```

Fig. 2. The algorithm to compute $\text{SEB}(S)$.

Lemma 4. *Let T and c as in Lemma 3 and let $p \in B(c, T)$ lie behind $\text{aff}(T)$, precisely,*

$$\langle p - c, \text{CC}(T) - c \rangle \geq \langle \text{CC}(T) - c, \text{CC}(T) - c \rangle. \quad (1)$$

Then p is contained in $B(c', T)$ for any $c' \in [c, \text{CC}(T)]$.

It remains to identify which point of the boundary set T should be dropped in case that $c \in \text{aff}(T)$ but $c \notin \text{conv}(T)$. Here are the suitable candidates.

Lemma 5. *Let T and c as in Lemma 3 and assume that $c \in \text{aff}(T)$. Let*

$$c = \sum_{q \in T} \lambda_q q, \quad \sum_{q \in T} \lambda_q = 1 \quad (2)$$

be the affine representation of c with respect to T . If $c \notin \text{conv}(T)$ then $\lambda_p < 0$ for at least one $p \in T$ and any such p satisfies inequality (1) with T replaced by the reduced set $T \setminus \{p\}$ there.

Combining Lemmata 4 and 5, we see that if we drop a point with negative coefficient in (2), this point will not stop us in the subsequent walking step.

The Algorithm in detail. Fig. 2 gives a formal description of our algorithm. The correctness follows easily from the previous considerations and we will address the issue of termination soon. Before, let us consider an example in the plane. Figure 3, (a)–(c), depicts all three iterations of our algorithm on a four-point

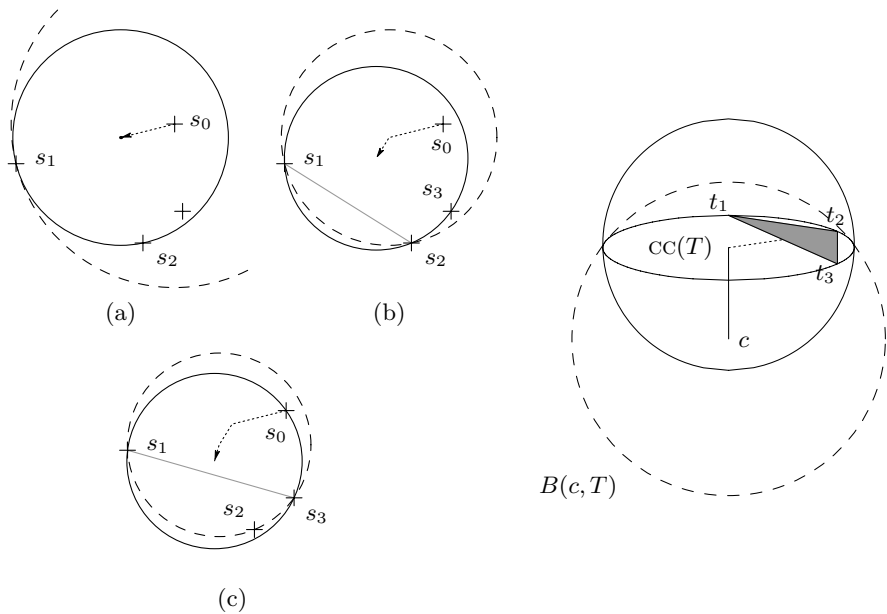


Fig. 3. A full run of the algorithm in 2D (left) and two consecutive steps in 3D (right).

set. Each picture shows the current ball $B(c, T)$ just before (dashed) and right after (solid) the walking phase.

After the initialization $c = s_0$, $T = \{s_1\}$, we move towards the singleton T until s_2 hits the boundary (step (a)). The subsequent motion towards the circumcenter of two points is stopped by the point s_3 , yielding a 3-element support (step (b)). Before the next walking we drop the point s_2 from T . The last movement (c) is eventually stopped by s_0 and then the center lies in the convex hull of $T = \{s_0, s_1, s_3\}$.

Observe that the 2-dimensional case obscures the fact that in higher dimensions, the target $\text{CC}(T)$ of a walk need not lie in the convex hull of the support set T . In the right picture of Fig. 3, the current center c first moves to $\text{CC}(T) \notin \text{conv}(T)$, where $T = \{t_1, t_2, t_3\}$. Then, t_2 is dropped and the walk continues towards $\text{aff}(T \setminus \{t_2\})$.

Termination. It is not clear whether the algorithm as stated in Fig. 2 always terminates. Although the radius of the ball clearly decreases whenever the center moves, it might happen that a stopper already lies on the current ball and thus no real movement is possible. In principle, this might happen repeatedly from some point on, i.e., we might run in an infinite cycle, perpetually collecting and dropping points without ever moving the center at all. However, for points in sufficiently general position such infinite loops cannot occur.

Proposition 1. *If for all affinely independent subsets $T \subseteq S$, no point of $S \setminus T$ lies on the circumsphere of T then algorithm $\text{seb}(S)$ terminates.*

Proof. Right after a dropping phase, the dropped point cannot be reinserted (Lemmata 4 and 5) and by assumption no other point lies on the current boundary. Thus, the sequence of radii measured right before the dropping steps is strictly decreasing; and since at least one out of d consecutive iterations demands a drop, it would have to take infinitely many values if the algorithm did not terminate. But this is impossible because before a drop, the center c coincides with the circumcenter $cc(T)$ of one out of finitely many subsets T of S . $\square$

The degenerate case. In order to achieve termination for arbitrary instances, we equip the procedure $seb(S)$ with the following simple rule, resembling Bland's pivoting rule for the simplex algorithm [13] (for simplicity, we will actually call it *Bland's rule* in the sequel):

Fix an arbitrary order on the set S . When dropping a point with negative coefficient in (2), choose the one of smallest rank in the order. Also, pick the smallest-rank point for inclusion in T when the algorithm is simultaneously stopped by more than one point during the walking phase.

As it turns out, this rule prevents the algorithm from “cycling”, i.e., it guarantees that the center of the current ball cannot stay at its position for an infinite number of iterations.

Theorem 1. *Using Bland's rule, $seb(S)$ terminates.*

Proof. Assume for a contradiction that the algorithm cycles, i.e., there is a sequence of iterations where the first support set equals the last and the center does not move. We assume w.l.o.g. that the center coincides with the origin. Let $C \subseteq S$ denote the set of all points that enter and leave the support during the cycle and let among these be m the one of maximal rank.

The key idea is to consider a slightly modified instance X of the SEB problem. Choose a support set $D \not\ni m$ right after dropping m and let $X := D \cup \{-m\}$, mirroring the point m at 0. There is a unique affine representation of the center 0 by the points in $D \cup \{m\}$, where by Bland's rule, the coefficients of points in D are all nonnegative while m 's is negative. This gives us a *convex* representation of 0 by the points in X and we may write

$$0 = \left\langle \sum_{p \in X} \lambda_p p, cc(I) \right\rangle = \sum_{p \in D} \lambda_p \langle p, cc(I) \rangle - \lambda_{-m} \langle m, cc(I) \rangle. \quad (3)$$

We have introduced the scalar products because of their close relation to criterion (1) of the algorithm. We bound these by considering a support set $I \not\ni m$ just before insertion of the point m . We have $\langle m, cc(I) \rangle < \langle cc(I), cc(I) \rangle$ and by Bland's rule and the maximality of m , there cannot be any other points of C in front of $\text{aff}(I)$; further, all points of D that do not lie in C must, by definition, also lie in I . Hence, we get $\langle p, cc(I) \rangle \geq \langle cc(I), cc(I) \rangle$ for all $p \in I$. Plugging these inequalities into (3) we obtain

$$0 > \left(\sum_{p \in D} \lambda_p - \lambda_{-m} \right) \langle cc(I), cc(I) \rangle = (1 - 2\lambda_{-m}) \langle cc(I), cc(I) \rangle,$$

which implies $\lambda_{-m} > 1/2$, a contradiction to Lemma 2. $\square$

4 The Implementation

We have programmed our algorithm in C++ using floating point arithmetic. At the heart of this implementation is a dynamic QR -decomposition, which allows updates for point insertion into and deletion from T and supports the two core operations of our algorithm:

- compute the affine coefficients of some $p \in \text{aff}(T)$,
- compute the circumcenter $\text{CC}(T)$ of T .

Our implementation, however, does not tackle the latter task in the present formulation. From part (i) of Lemma 3 we know that the circumcenter of T coincides with the orthogonal projection of c onto $\text{aff}(T)$ and this is how we actually compute $\text{CC}(T)$ in practice. Using this reformulation, we shall see that the two tasks at hand are essentially the same problem. We briefly sketch the main ideas behind our implementation, which are combinations of standard concepts from linear algebra.

Computing orthogonal projections and affine coefficients. In order to apply linear algebra, we switch from the affine space to a linear space by fixing some “origin” q_0 of $T = \{q_0, q_1, \dots, q_r\}$ and defining the relative vectors $a_i = q_i - q_0$, $1 \leq i \leq r$. For the matrix $A = [a_1, \dots, a_r]$, which has full column rank r , we maintain a QR -decomposition $QR = A$, that is, an orthogonal $d \times d$ matrix Q and a rectangular $d \times r$ matrix $R = \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}$, where $\hat{R}$ is square upper triangular.

Recall how such a decomposition can be used to “solve” an overdetermined system of linear equations $Ax = b$ (the right hand side $b \in \mathbb{R}^d$ being given) [14, Sec. 5.3]: Using orthogonality of Q , first compute $y := Q^{-1}b = Q^T b$; then discard the lower $d - r$ entries of $y = \begin{bmatrix} \hat{y} \\ * \end{bmatrix}$ and evaluate $\hat{R}x^* = \hat{y}$ through back substitution. The resulting x^* is known to minimize the residual $\|Ax - b\|$, which means that Ax^* is the unique point in $\text{im}(A)$ closest to b . In other words, Ax^* is the orthogonal projection of b onto $\text{im}(A)$.

This already solves both our tasks. The affine coefficients of some point $p \in \text{aff}(T)$ are exactly the entries of the approximation x^* for the shifted equation $Ax = p - q_0$ (the missing coefficient of q_0 follows directly from the others); further, for arbitrary p , Ax^* is just the orthogonal projection of p onto $\text{aff}(T)$. With a QR -decomposition of A at hand, these calculations can be done in quadratic time (in the dimension) as seen above.

The computation of orthogonal projections even allows some improvement. By reformulating it as a Gram-Schmidt-like procedure and exploiting a duality in the QR -decomposition, we can obtain running times of order $\min\{\text{rank}(A), d - \text{rank}(A)\} \cdot d$. This should, however, be seen only as a minor technical modification of the general procedure described above.

Maintaining the QR -decomposition. Of course, computing orthogonal projections and affine coefficients in quadratic time would not be of much use if we had to set up a complete QR -decomposition of A in cubic time each time a point

is inserted into or removed from the support set T —the basic modifications applied to T in each iteration.

Golub and van Loan [14, Sec. 12.5] describe how to update a QR -decomposition in quadratic time, using *Givens rotations*. We briefly present those techniques and show how they apply to our setting of points in affine space.

Adding a point p to T corresponds to appending the column vector $u = p - q_0$ to A yielding a matrix A' of rank $r + 1$. To incorporate this change into Q and R , append the column vector $w = Q^T u$ to R so that the resulting matrix R' satisfies the desired equation $QR' = A'$. But then R' is no longer in upper triangular form. This defect can be repaired by application of $d - r - 1$ Givens rotations $G_{r+1}, \dots, G_{d-1}$ from the left (yielding the upper triangular matrix $R'' = G_{r+1} \cdots G_{d-1} R'$). Applying the transposes of these orthogonal matrices to Q from the right (giving the orthogonal matrix $Q' = Q G_{d-1}^T \cdots G_{r+1}^T$) then provides consistency again ($Q' R'' = A'$). Since multiplication with a Givens rotator effects only two columns of Q , the overall update takes $\mathcal{O}(d^2)$ steps.

Removing a point from T works similarly to insertion. Simply erase the corresponding column from R and shift the higher columns one place left. This introduces one subdiagonal entry in each shifted column which again can be zeroed with a linear number of Givens rotations, resulting in a total number of $\mathcal{O}(dk)$ steps, where $k < d$ is the number of columns shifted.

The remaining task of removing the origin q_0 from T (which does not work in the above fashion since q_0 does not correspond to a particular column of A) can also be dealt with efficiently (using an appropriate rank-1-update). Thus, all updates can be realized in quadratic time. (Observe that we used the matrices A and A' only for explanatory purposes. Our program does not need to store them explicitly but performs all computation on Q and R directly.)

Stability, termination, and verification. As for numerical stability, QR -decomposition itself behaves nicely since all updates on Q and R are via orthogonal Givens rotators [14, Sec. 5.1.10]. However, for our coefficient computations and orthogonal projections to work, we have to avoid degeneracy of the matrix R . Though in theory this is guaranteed through the affine independence of T , we introduce a stability threshold in our floating point implementation to protect against unfortunate interaction of rounding errors and geometric degeneracy in the input set. This is done by allowing only such stopping points to enter the support set that are not closer to the current affine hull of T than some ϵ . (Remember that points behind $\text{aff}(T)$ are ignored anyway because of Lemma 4.)

Our code is equipped with a result checker, which upon termination verifies whether all points of S really lie in the computed ball and whether the support points all lie on the boundary. We further do some consistency checking on the final QR -decomposition by determining the affine coefficients of each individual point in T . In all our test runs, the overall error computed thus was never larger than 10^{-12} , about 10^4 times the machine precision.

Finally, we note that while Bland's rule guarantees termination in theory, it is slow in practice. As with LP-solvers, we resort to a different heuristic in practice, which yields very satisfactory running-time behavior and has the fur-

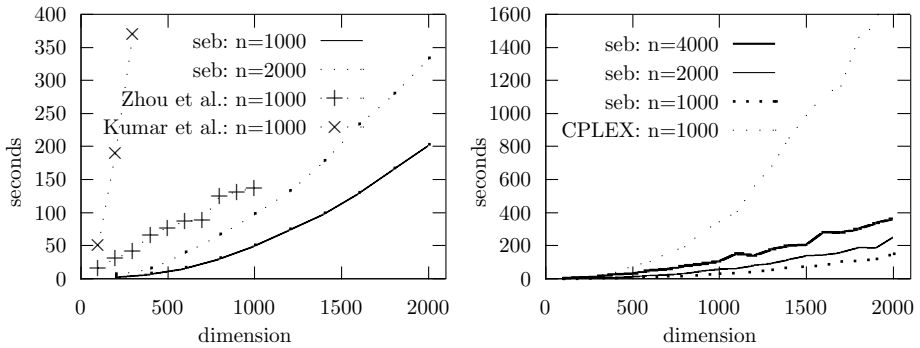


Fig. 4. (a) Our Algorithm seb, Zhou et al., and Kumar et al. on uniform distribution, (b) Algorithm seb on normal distribution.

ther advantage of greater robustness with respect to roundoff errors: The rule for deletion from T in the dropping phase is to pick the point t of minimal λ_t in (2). For insertion into T in the walking phase we consider, roughly speaking, all points of S which would result in almost the same walking distance as the actual stopping point p (allowing only some additional ϵ) and choose amongst these the one farthest from $\text{aff}(T)$.

Note that our stability threshold and the above selection threshold are a concept entirely different from the approximation thresholds of [9] and [11]. Our ϵ 's do not enter the running time and choosing them close to machine precision already resulted in very stable behavior in practice.

5 Testing Results

We have run the floating point implementation of our algorithm on random point sets drawn from different distributions:

- uniform distribution in the unit cube,
- normal distribution with standard deviation 1,
- uniform distribution on the surface of the unit sphere, with each point perturbed in direction of the center by some number drawn uniformly at random from a small interval $[-\delta, +\delta]$.

The tests were performed on a 480Mhz Sun Ultra 4 workstation.

Figure 4(a) shows the running time of our algorithm on instances with 1,000 and 2,000 points drawn uniformly at random from the unit cube in up to 2,000 dimensions. For reference, we included the running times given in [9] and [11] on similar point sets. However, these data should be interpreted with care: they are taken from the respective publications and we have not recomputed them under our conditions (the code of [11] is not available). However, the hardware was comparable to ours. Also observe that Zhou et al. implemented their algorithm

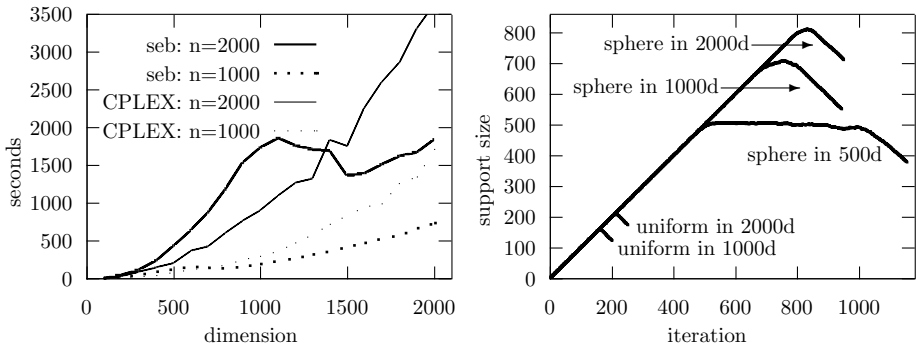


Fig. 5. (a) Algorithm seb and CPLEX on almost spherical distribution ($\delta = 10^{-4}$), (b) Support-size development depending on the number of iterations, for 1,000 points in different dimensions, distributed uniformly and almost spherically ($\delta = 10^{-3}$)

in Matlab, which is not really comparable with our C++ implementation. Still, the figures give an idea of the relative performances of the three methods for several hundred dimensions.

On the normal distribution our algorithm performs similarly as for the uniform distribution. Figure 4(b) contains plots for sets of 1,000, 2,000, and 4,000 points. We compared these results to the performance of the general-purpose QP-solver of CPLEX (version 6.6.0, which is the latest version available to us). We outperform CPLEX by wide margins. (The running times of CPLEX on 2,000 and 4,000 points, which are not included in the figure, scale almost uniformly by a factor of 2 resp. 4 on the tested data.) Again, these results are to be seen in the proper perspective; it is of course not surprising that a dedicated algorithm is superior to a general-purpose code; moreover, the QP arising from the SEB problem are not the “typical” QP that CPLEX is tuned for. Still, the comparison is necessary in order to argue that off-the-shelf methods cannot successfully compete with our approach.

The most difficult inputs for our code are sets of (almost) cospherical points. In such situations, it typically happens that many points enter the support set in intermediate steps only to be dropped again later. This is quite different to the case of the normal distribution, where a large fraction of the points will never enter the support and the algorithm needs much fewer iterations. Figure 5 compares our algorithm and again CPLEX on almost-spherical instances. For smaller dimensions, CPLEX is faster, but starting from roughly $d = 1,500$, we again win.

The papers of Zhou et al. as well as Kumar et al. do not contain tests with almost-cospherical points. In case of the QP-solver of CPLEX, our observation is that the point distribution has no major influence on the runtime; in particular, cospherical points do not give rise to particularly difficult inputs. It might be the case that the same is true for the methods of Zhou et al. and Kumar et al., but it would still be interesting to verify this on concrete examples.

To provide more insight into the actual behavior of our algorithm, Figure 5(b) shows the support-size development for complete runs on different inputs. In all our tests we observed that the computation starts with a long point-collection phase during which no dropping occurs. This initial phase is followed by an intermediate period of dropping and inserting during which the support size changes only very little. Finally, there is a short dropping phase almost without new insertions. The intermediate phase is usually quite short, except for almost spherical distributions with n considerably larger than d . The explanation for this phenomenon being that only for such distributions there are many candidate points for the final support set which are repeatedly dropped and inserted several times. With growing dimension, more and more points of an almost-spherical distribution belong into the final support set, leaving only few points ever to be dropped. We thank Emo Welzl for pointing out this fact, thus explaining the nonmonotone running-time behavior of our algorithm in Fig. 5(a).

References

1. Megiddo, N.: Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.* **12** (1983) 759–776
2. Dyer, M.E.: A class of convex programs with applications to computational geometry. In: *Proc. 8th Annu. ACM Sympos. Comput. Geom.* (1992) 9–15
3. Welzl, E.: Smallest enclosing disks (balls and ellipsoids). In Maurer, H., ed.: *New Results and New Trends in Computer Science*. Volume 555 of *Lecture Notes Comput. Sci.* Springer-Verlag (1991) 359–370
4. Gärtner, B.: Fast and robust smallest enclosing balls. In: *Proc. 7th Annual European Symposium on Algorithms (ESA)*. Volume 1643 of *Lecture Notes Comput. Sci.*, Springer-Verlag (1999) 325–338
5. Gärtner, B., Schönherr, S.: An efficient, exact, and generic quadratic programming solver for geometric optimization. In: *Proc. 16th Annu. ACM Sympos. Comput. Geom.* (2000) 110–118
6. Ben-Hur, A., Horn, D., Siegelmann, H.T., Vapnik, V.: Support vector clustering. *Journal of Machine Learning Research* **2** (2001) 125–137
7. Bulatov, Y., Jambawalikar, S., Kumar, P., Sethia, S.: Hand recognition using geometric classifiers (2002) Abstract of presentation for the DIMACS Workshop on Computational Geometry (Rutgers University).
8. Goel, A., Indyk, P., Varadarajan, K.R.: Reductions among high dimensional proximity problems. In: *Symposium on Discrete Algorithms*. (2001) 769–778
9. Kumar, P., Mitchell, J.S.B., Yıldırım, E.A.: Computing core-sets and approximate smallest enclosing hyperspheres in high dimensions (2003) To appear in the *Proceedings of ALENEX'03*.
10. ILOG, Inc.: *ILOG CPLEX 6.5 user's manual* (1999)
11. Zhou, G., Toh, K.C., Sun, J.: Efficient algorithms for the smallest enclosing ball problem. *Manuscript* (2002)
12. Hopp, T.H., Reeve, C.P.: An algorithm for computing the minimum covering sphere in any dimension. *Technical Report NISTIR 5831*, National Institute of Standards and Technology (1996)
13. Chvátal, V.: *Linear programming*. W. H. Freeman, New York, NY (1983)
14. Golub, G.H., van Loan, C.F.: *Matrix Computations*. third edn. Johns Hopkins University Press (1996)

Automated Generation of Search Tree Algorithms for Graph Modification Problems

Jens Gramm*, Jiong Guo**, Falk Hüffner, and Rolf Niedermeier**

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen, Germany

{`gramm, guo, hueffner, niederm`}@informatik.uni-tuebingen.de

Abstract. We present a (seemingly first) framework for an automated generation of exact search tree algorithms for NP-hard problems. The purpose of our approach is two-fold—rapid development and improved upper bounds. Many search tree algorithms for various problems in the literature are based on complicated case distinctions. Our approach may lead to a much simpler process of developing and analyzing these algorithms. Moreover, using the sheer computing power of machines it may also lead to improved upper bounds on search tree sizes (i.e., faster exact solving algorithms) in comparison with previously developed “hand-made” search trees.

1 Introduction

In the field of exactly solving NP-hard problems, almost always the developed algorithms employ exhaustive search based on a clever *search tree* (also called *splitting*) strategy. For instance, search tree based algorithms have been developed for SATISFIABILITY [6], MAXIMUM SATISFIABILITY [1], EXACT SATISFIABILITY [4], INDEPENDENT SET [3,12], VERTEX COVER [2,11], and 3-HITTING SET [10]. Moreover, most of these problems have undergone some kind of “evolution” towards better and better exponential-time algorithms. The improved upper bounds on the running times, however, usually are at the cost of distinguishing between more and more combinatorial cases which makes the development and the correctness proofs an awesome and error-prone task. For example, in a series of papers the upper bound on the search tree size for an algorithm solving MAXIMUM SATISFIABILITY was improved from 1.62^K to 1.38^K to 1.34^K to recently 1.32^K [1], where K denotes the number of clauses in the given formula in conjunctive normal form. In this paper, seemingly for the first time, we present an automated approach for the development of efficient search tree algorithms, focusing on NP-hard graph modification problems.

* Supported by the Deutsche Forschungsgemeinschaft (DFG), research project OPAL (optimal solutions for hard problems in computational biology), NI 369/2.

** Supported by the Deutsche Forschungsgemeinschaft (DFG), junior research group PIAF (fixed-parameter algorithms), NI 369/4.

Our approach is based on the separation of two tasks in the development of search tree algorithms—namely, the investigation and development of clever “problem-specific rules” (this is usually the creative, thus, the “human part”), and the analysis of numerous cases using these problem-specific rules (this is the “machine part”). The software environment we deliver can also be used in an interactive way in the sense that it points the user to the worst case in the current case analysis. Then, the user may think of additional problem-specific rules to improve this situation, obtain a better bound, and repeat this process.

The automated generation of search tree algorithms in this paper is restricted to the class of graph modification problems [7,9], although the basic ideas appear to be generalizable to other graph and even non-graph problems. In particular, we study the following NP-complete edge modification problem CLUSTER EDITING, which is motivated by data clustering applications in computational biology [13]:

Input: An undirected graph $G = (V, E)$, and a nonnegative integer k .

Question: Can we transform G , by deleting and adding at most k edges, into a graph that consists of a disjoint union of cliques?

In [5] we gave a search tree algorithm solving CLUSTER EDITING in $O(2.27^k + |V|^3)$ time. This algorithm is based on case distinctions developed by “human case analysis” and it took us about three months of development and verification. Now, based on some relatively simple problem-specific rules, we obtained an $O(1.92^k + |V|^3)$ time algorithm for the same problem in about one week.

The example application to CLUSTER EDITING exhibits the power of our approach, whose two main potential benefits we see as rapid development and improved upper bounds due to automation of tiresome and more or less schematic but extensive case-by-case analysis. Besides CLUSTER EDITING, we present applications of our approach to other NP-complete graph modification problems.

Due to the lack of space some details are deferred to the full paper.

2 Preliminaries

We only deal with undirected graphs $G = (V, E)$. By $N(v) := \{u \mid \{u, v\} \in E\}$ we denote the *neighborhood* of $v \in V$. We call a graph $G' = (V', E')$ *vertex-induced subgraph* of graph $G = (V, E)$ iff $V' \subseteq V$ and $E' = \{\{u, v\} \mid u, v \in V' \text{ and } \{u, v\} \in E\}$. A *graph property* is simply a mapping from the set of graphs onto true and false. Our core problem GRAPH MODIFICATION is as follows.

Input: Graph G , a graph property Π , and a nonnegative integer k .

Question: Is there a graph G' such that $\Pi(G')$ holds and such that we can transform G into G' by altogether at most k edge additions, edge deletions, and vertex deletions?

In this paper, we deal with special cases of GRAPH MODIFICATION named EDGE MODIFICATION (only edge additions and deletions are allowed), EDGE DELETION (only edge deletions allowed), and VERTEX DELETION (only vertex deletions allowed). The concrete applications of our framework to be presented here refer to properties Π that have a *forbidden subgraph characterization*. For

instance, consider CLUSTER EDITING. Here, the property Π is “to consist of a disjoint union of cliques.” It holds that this Π is true for a graph G iff G has no P_3 (i.e., a path consisting of three vertices) as a vertex-induced subgraph. The corresponding EDGE DELETION problem is called CLUSTER DELETION.

Search tree algorithms. Perhaps the most natural way to organize exhaustive search is to use a search tree. For instance, consider the NP-complete VERTEX COVER problem where, given a graph $G = (V, E)$ and a positive integer k , the question is whether there is a set of vertices $C \subseteq V$ with $|C| \leq k$ such that each edge in E has at least one of its two endpoints in C . For an arbitrary edge $\{u, v\}$, at least one of the vertices u and v has to be in C . Thus, we can branch the recursive search into two cases, namely $u \in C$ or $v \in C$. Since we are looking for a set C of size at most k we easily obtain a search tree of size $O(2^k)$.

Analysis of search tree sizes. If the algorithm solves a problem of “size” s and calls itself recursively for problems of “sizes” $s - d_1, \dots, s - d_i$, then $(d_1, \dots, d_i)$ is called the *branching vector* of this recursion. It corresponds to the recurrence $t_s = t_{s-d_1} + \dots + t_{s-d_i}$, with $t_i = 1$ for $0 \leq i < d$ and $d = \max\{d_1, \dots, d_i\}$, (to simplify matters, without any harm, we only count the number of leaves here) and its characteristic polynomial $z^d = z^{d-d_1} + \dots + z^{d-d_i}$. We often refer to the case distinction corresponding to a branching vector $(d_1, \dots, d_i)$ as $(d_1, \dots, d_i)$ -*branching*. The characteristic polynomial as given here has a unique positive real root α and $t_s = O(\alpha^s)$. We call α the *branching number* that corresponds to the branching vector $(d_1, \dots, d_i)$.

In our framework an often occurring task is to “concatenate” *branching vectors*. For example, consider the two branching vector sets $S_1 = \{(1, 2), (1, 3, 3)\}$ and $S_2 = \{(1)\}$. We have to determine the best branching vector when concatenating every element from S_1 with every element from S_2 . We cannot simply take the best branching vector from S_1 and concatenate it with the best one from S_2 . In our example, branching vector $(1, 2)$ (branching number 1.62) is better than branching vector $(1, 3, 3)$ (branching number 1.70), whereas with respect to concatenation $(1, 2, 1)$ (branching number 2.42) is worse than $(1, 3, 3, 1)$ (branching number 2.36). Since in our applications the sets S_1 and S_2 can generally get rather large, it would save much time not having to check every pair of combinations. We use the following simplification. Consider a branching vector as a multi-set, i.e., identical elements may occur several times but the order of the elements plays no role. Then, comparing two branching vectors b_1 and b_2 , we say that b_2 is *subsumed by* b_1 if there is an injective mapping f of elements from b_1 onto elements from b_2 such that for every $x \in b_1$ it holds that $x \geq f(x)$. Then, if one branching vector is subsumed by another one from the same set, the subsumed one can be discarded from further consideration.

3 The General Technique

Search tree algorithms basically consist of a set of branching rules. Branching rules are usually based on local substructures, e.g., for graph problems, on induced subgraphs having up to s vertices for a constant integer s ; we refer to

graphs having s vertices as *size- s graphs*. Then, each branching rule specifies the branching for each particular local substructure. The idea behind our automation approach is roughly described as follows:

(1) For constant s , enumerate all “relevant” local substructures of size s such that every input instance of the given graph problem has s vertices inducing at least one of the enumerated local substructures.

(2) For every local substructure enumerated in Step (1), check all possible branching rules for this local substructure and select the one corresponding to the *best*, i.e. smallest, branching number. The set of all these best branching rules then defines our search tree algorithm.

(3) Determine the worst-case branching rule among the branching rules stored in Step (2).

Note that both in Step (1) and Step (2), we usually make use of further *problem-specific rules*: For example, in Step (1), problem-specific rules can determine input instances which do not need to be considered in our enumeration, e.g., instances which can be solved in polynomial time, which can be simplified due to reduction rules, etc.

In the next two subsections, we discuss Steps (1) and (2), respectively, in more detail. We will use CLUSTER DELETION as a running example.

3.1 Computing a Branching Rule for a Local Subgraph

We outline a general framework to generate, given a size- s graph $G_s = (V_s, E_s)$ ¹ for constant s , an “optimal” branching rule for G_s . To compute a search tree branching rule, we, again, use a search tree to explore the space of possible branching rules. This search tree is referred to as *meta search tree*. We describe our framework for the example of CLUSTER DELETION. Our central reference point in this subsection is the meta search tree procedure `compute_br()` given in Fig. 1. In the following paragraphs we describe `compute_br()` in a step-by-step manner.

(1) Branching rules and branching objects. A branching rule for G_s specifies a set of “simplified” (to be made precise in the next paragraph) graphs $G_{s,1}, G_{s,2}, \dots, G_{s,r}$. When invoking the branching rule, one would replace, for every $G_{s,i}$, $1 \leq i \leq r$, G_s by $G_{s,i}$ and invoke the search tree procedure recursively on the thereby generated instances. By definition, the branching rule has to satisfy the following property: a “solution” is an optimal solution for G_s iff it is “best” among the optimal solutions for all $G_{s,i}$, $1 \leq i \leq r$. This is referred to by saying that the branching rule is *complete*. The *branching objects* are the objects on which the branching rule to be constructed branches. In CLUSTER DELETION, the *branching objects* are the vertex pairs of the input graph G_s since we obtain a solution graph by deleting edges.

(2) Annotations. A “simplified” graph $G_{s,i}$, $1 \leq i \leq r$, is obtained from G_s by assigning labels to a subset of branching objects in G_s . The employed set of labels is problem-specific. Depending on the problem, certain branching objects

¹ We assume that the vertices of the graph are ordered.

Procedure `compute_br`(π)

Global: Graph $G_s = (V_s, E_s)$.

Input: Annotation π for G_s (for a definition of annotations see paragraph (2)).

Output: Set B of branching rules for G_s with annotation π .

Method:

```

 $B := \emptyset;$                                 /* set of branching rules, to be computed */
 $\pi := \text{br\_reduce}(\pi);$                       /* paragraph (4) */
for all  $\{u, v\} \in E_s$  with  $u < v$  and  $\pi(u, v) = \text{undef}$  do
     $\pi_1 := \pi;$ 
     $\pi_1(u, v) := \text{permanent};$                 /* annotate edge as permanent */
     $B_1 := \text{compute\_br}(\pi_1);$ 
     $\pi_2 := \pi;$ 
     $\pi_2(u, v) := \text{forbidden};$                 /* annotate edge as forbidden */
     $B_2 := \text{compute\_br}(\pi_2);$ 
    /* concatenating and filtering branching rules */
     $B := B \cup \text{br\_concatenate}(B_1, B_2);$ 
endfor;
if  $\pi$  implies edge deletions for  $G_s$  then  $B := B \cup \{\pi\}$  endif;
return  $B$ ;
```

Fig. 1. Meta search tree procedure for CLUSTER DELETION in pseudocode

may also initially carry “problem-inherent” labels which cannot be modified by the meta search tree procedure. An *annotation* is a partial mapping π from the branching objects to the set of labels; if no label is assigned to a branching object then π maps to “undef.” Let π and π' both be annotations for G_s , then π' *refines* π iff, for every branching object b , it holds that $\pi(b) \neq \text{undef} \Rightarrow \pi'(b) = \pi(b)$.

As to CLUSTER DELETION, the labels for a vertex pair $u, v \in V_s$ can be chosen as *permanent* (i.e., the edge is in the solution graph to be constructed) or *forbidden* (i.e., the edge is not in the solution graph to be constructed). In CLUSTER DELETION, all vertex pairs sharing no edge are initially assigned the label *forbidden* since edges cannot be added; these are the problem-inherent labels. By G_s with annotation π , we, then, refer to the graph obtained from G_s by deleting $\{u, v\} \in E_s$ if π assigns the label *forbidden* to (u, v) . In this way, an annotation can be used to specify one branch of a branching rule.

(3) Representation of branching rules. A branching rule for G_s with annotation π can be represented by a set A of annotations for G_s such that, for every $\pi' \in A$, π' refines π . Then, every $\pi' \in A$ specifies one branch of the branching rule. A set A of annotations has to satisfy the following three conditions: (a) The branching rule is complete. (b) Every annotation decreases the *search tree measure*, i.e., the parameter with respect to which we intend to measure the search tree size. (c) The subgraph consisting of the annotated branching objects has to fulfill every property required for a solution of the considered graph problem.

In CLUSTER DELETION, condition (b) implies that every annotation deletes at least one edge from the graph. Condition (c) means that the annotated vertex pairs do not form a P_3 , i.e., there are no $u, v, w \in V_s$ with $\pi(u, v) = \pi(v, w) = \text{permanent}$ and $\pi(u, w) = \text{forbidden}$.

(4) Problem-specific rules that refine annotations. To obtain non-trivial bounds it is decisive to have a set of problem-specific *reduction rules*. A reduction rule specifies how to refine a given annotation π to π' such that an optimal solution for the input graph with annotation π' is also an optimal solution for the input graph with annotation π .

For CLUSTER DELETION, we have the following reduction rule (for details see Sect. 4): Given a graph $G = (V, E)$ with annotation π , if there are three pairwise distinct vertices $u, v, w \in V$ with $\pi(u, v) = \pi(v, w) = \text{permanent}$, then we can replace π by an annotation π' which refines π by setting $\pi'(u, w)$ to permanent. Analogously, if $\pi(u, v) = \text{permanent}$ and $\pi(v, w) = \text{forbidden}$, then $\pi'(u, w) := \text{forbidden}$. In Fig. 1, the reduction rule is implemented by procedure `br_reduce( $\pi$ )`.

(5) Meta search tree, given in Fig. 1. The root of the meta search tree is, for a non-annotated input graph, given by calling `compute_br( $\pi_0$ )` with G_s and an annotation π_0 which assigns the problem-inherent labels to branching objects (e.g., in CLUSTER DELETION the forbidden labels to vertex pairs sharing no edge) and, apart from that, maps everything to `undef`. The call `compute_br( $\pi_0$ )` results in a set B of branching rules. From B , we select the *best* branching rule (smallest branching number).

In Fig. 2, we illustrate the meta search tree generated for CLUSTER DELETION and a size-4 graph.

(6) Storing already computed branching rules. To avoid processing the same annotations several times, we store an annotation which has already been processed together with its computed set of branching rules.

(7) Generalizing the framework. In this section, we concentrated on the CLUSTER DELETION problem. We claim, however, that this framework is usable for graph problems in general. Two main issues where changes have to be made depending on the considered problem are given as follows:

- (a) In CLUSTER DELETION, the branching objects are vertex pairs and the possible labels are “permanent” and “forbidden.” In general, the branching objects and an appropriate set of labels for them are determined by the considered graph problem. For example, in VERTEX COVER, the objects to branch on are vertices and, thus, the labels would be assigned to the vertices. The labels would be, e.g., “is in the vertex cover” and “is not in the vertex cover.”
- (b) The reduction rules are problem-specific. To design an appropriate set of reduction rules working on local substructures is the most challenging part when applying our framework to a new problem.

In this subsection, we presented the meta search tree procedure for the example of CLUSTER DELETION. As input it takes a local substructure and a set of reduction rules. It exhaustively explores the set of *all possible* branching rules on this local substructure, taking into account the given reduction rules.

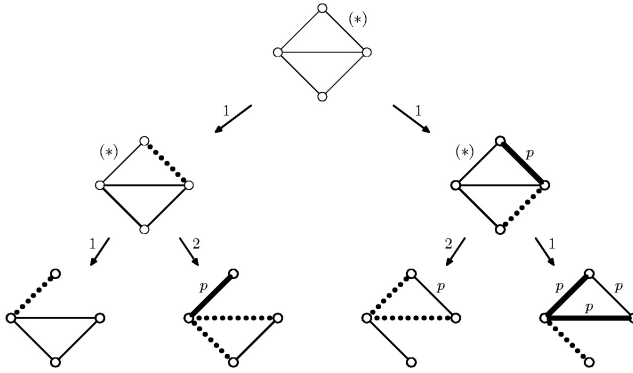


Fig. 2. Illustration of a meta search tree traversal for CLUSTER DELETION. At the root we have a size-4 input graph having no labels. *Arrows* indicate the branching steps of the meta search tree. We only display branches of the meta search tree which contribute to the computed branching rule. The vertex pair on which we branch is indicated by (*). Permanent edges are indicated by *p*, vertex pairs sharing no edge are implicitly forbidden (*bold* or *dotted lines* indicate when a vertex pair is newly set to permanent or forbidden, respectively). Besides the vertex pair on which we branch, additional vertex pairs are set to permanent or forbidden due to the problem-specific reduction rule explained in Sect. 3.1(4). The numbers at the arrows indicate the number of edges deleted in the respective branching step. The resulting branching rule is determined by the leaves of this tree and the corresponding branching vector is (2, 3, 3, 2)

This yields the following theorem where π_0 denotes the annotation assigning the problem-inherent labels of CLUSTER DELETION:

Theorem 1. *Given a graph G_s with s vertices for constant s , the set of branching rules returned by `compute_br`(π_0) (Fig. 1) contains, for this fixed s , an optimal branching rule for CLUSTER DELETION that can be obtained by branching only on vertex pairs from G_s and by using the reduction rules performed by `br_reduce`().*

3.2 A More Sophisticated Enumeration of Local Substructures

Using problem-specific rules, we can improve our enumeration of local substructures (here, graphs of size s for constant s) as indicated in Step (1) of our automation approach described in the introducing part of Section 3. To this end, we can, firstly, decrease the number of graphs that have to be enumerated and, secondly, we can, in this way, improve the worst-case branching by excluding “unfavorable” graphs from the enumeration (details will be given in the full paper). We start the enumeration with small graphs, *expanding* them recursively to larger graphs. Problem-specific rules allow us to compute non-trivial ways to expand a given graph, thereby improving the worst-case branching in the resulting algorithm. Since we start this expansion with small graphs, we can employ

a user-specified “cut-off value” and save to expand a graph as soon as it yields a branching number better than this value; this further accelerates our technique.

4 Applications to Graph Modification Problems

The developed software consists of about 1900 lines of Objective Caml code and 1500 lines of low-level C code for the graph representation, which uses simple bit vectors. The generation of canonical representations of graphs (for isomorphism tests and hash table operations) is done by the *nauty* library [8]. Branching vector sets are represented as tries, which allow for efficient implementation of the subsumption rules presented in Sect. 2. The tests were performed on a 2.26 GHz Pentium 4 PC with 1 GB memory running Linux. Memory requirements were up to 300 MB. We measured a variety of values:

- size:** Maximum number of vertices in the local subgraphs considered;
- time:** Total running time;
- isom:** Percentage of the time spent for the isomorphism tests;
- concat:** Percentage of the time spent for concatenating branching vector sets;
- graphs:** Number of graphs for which a branching rule was calculated;
- maxbn:** Maximum branching number of the computed set of branching rules (determining the worst-case bound of the resulting algorithm);
- avgbn:** Average branching number of the computed set of branching rules; assuming that every induced subgraph appears with the same likelihood, $(\text{avgbn})^k$ would give the average size of the employed search trees, where k is the number of graph modification operations.
- bvlen:** Maximum length of a branching vector occurring in the computed set of branching rules;
- medlen:** Median length of branching vectors occurring in the computed set of branching rules;
- maxlen:** Length of longest branching vector generated in a node of the meta search tree (including intermediary branching vectors);
- bvset:** Size of largest branching vector set in a node of the meta search tree.

4.1 CLUSTER EDITING

CLUSTER EDITING is NP-complete and has been used for the clustering of gene expression data [13]. In [5] we gave a fixed-parameter algorithm for this problem based on a bounded search tree of size $O(2.27^k)$.

Problem-specific rules. Following the general scenario from Sect. 3, we make use of problem-specific rules in our framework for an input graph $G = (V, E)$. We use the same labels “forbidden” and “permanent” as for CLUSTER DELETION.

Rule 1: While enumerating subgraphs, we consider only instances containing a P_3 as a vertex-induced subgraph.

Rule 2: For $u, v, w \in V$ such that both (u, v) and (v, w) are annotated as permanent, we annotate also pair (u, w) as permanent; if (u, v) is annotated as

Table 1. Results for CLUSTER EDITING: (1) Enumerating all size- s graphs containing a P_3 ; (2) Expansion scheme utilizing Proposition 1

	size	time	isom	concat	graphs	maxbn	avgbn	bvlen	medlen	maxlen	bvset
(1)	4	< 1 sec	3%	16%	5	2.42	2.33	5	5	8	7
(1)	5	2 sec	2%	50%	20	2.27	2.04	16	9	23	114
(1)	6	9 days	0%	100%	111	2.16	1.86	37	17	81	209179
(2)	4	< 1 sec	1%	20%	6	2.27	2.27	5	5	8	7
(2)	5	3 sec	0%	52%	26	2.03	1.97	16	12	23	114
(2)	6	9 days	0%	100%	137	1.92	1.80	37	24	81	209179

permanent and (v, w) as forbidden, then we annotate (u, w) as forbidden.

Rule 3: For every edge $\{u, v\} \in E$, we can assume that u and v have a common neighbor.

Rule 3 is based on the following proposition which allows us to apply a “good” branching rule if there is an edge whose endpoints have no common neighbor:

Proposition 1. *Given $G = (V, E)$. If there is an edge $\{u, v\} \in E$, where u and v have no common neighbor and $|(N(u) \cup N(v)) \setminus \{u, v\}| \geq 1$, then for CLUSTER EDITING a $(1, 2)$ -branching applies.* $\square$

Given a CLUSTER EDITING instance $(G = (V, E), k)$, we can apply the branching rule described in Proposition 1 as long as we find an edge satisfying the conditions of Proposition 1; the resulting graphs are called reduced with respect to Rule 3. If we already needed more than k edge modifications before the graph is reduced with respect to Rule 3 then we reject it.

Results and Discussion. See Table 1. Only using Rules 1 and 2, we obtain the worst-case branching number 2.16 when considering induced subgraphs containing six vertices. We observe a decrease in the computed worst-case branching number maxbn with every increase in the sizes of the considered subgraphs. The typical number of case distinctions for a subgraph (medlen) seems high compared to human-made case distinctions, but should pose no problem for an implementation.

When additionally using Rule 3, we use the expansion approach mentioned in Sect. 3.2. In this way, we can decrease maxbn to 1.92. This shows the usefulness of the expansion approach. It underlines the importance of devising a set of good problem-specific rules for the automated approach. Notably, the average branching number avgbn for the computed set of branching rules is significantly lower than the worst-case.

It can be observed from Table 1 that, for graphs with six vertices, the program spends almost all its running time on the concatenations of branching vectors; branching vector sets can contain huge amounts of incomparable branching vectors (bvset), and a single branching vector can get comparatively long (maxlen). Summarizing the results together with [5], we have the following theorem:

Theorem 2. CLUSTER EDITING can be solved in $O(1.92^k + |V|^3)$ time. $\square$

Table 2. Results for CLUSTER DELETION: (1) Enumerating all size- s graphs containing a P_3 ; (2) Expansion scheme

	size	time	isom	concat	graphs	maxbn	avgbn	bvlen	medlen	maxlen	bvset
(1)	4	< 1 sec	12%	12%	5	1.77	1.65	4	2	5	4
(1)	5	< 1 sec	37%	22%	20	1.63	1.52	8	2	13	83
(1)	6	6 min	4%	92%	111	1.62	1.43	16	2	35	7561
(2)	4	< 1 sec	7%	15%	6	1.77	1.70	4	2	5	4
(2)	5	< 1 sec	11%	33%	26	1.63	1.54	8	2	13	83
(2)	6	6 min	0%	97%	137	1.53	1.43	16	2	35	7561

Table 3. Results for CLUSTER VERTEX DELETION: (1) Enumerating all size- s graphs containing a P_3 ; (2) Expansion scheme with cutoff (see Sect. 3.2)

	size	time	isom	concat	graphs	maxbn	avgbn	bvlen	medlen	maxlen	bvset
(1)	6	1 sec	8%	12%	111	2.31	1.98	6	4	14	24
(1)	7	26 sec	19%	14%	852	2.27	1.86	6	4	21	65
(1)	8	39 min	34%	12%	11116	2.27	1.76	10	5	32	289
(2)	6	< 1 sec	0%	22%	74	2.31	2.06	6	4	13	12
(2)	7	< 1 sec	0%	27%	119	2.27	2.02	6	4	19	49
(2)	8	5 sec	0%	38%	205	2.27	2.00	8	4	25	146
(2)	9	46 sec	0%	53%	367	2.26	1.92	9	4	37	534
(2)	10	7 min	0%	69%	681	2.26	1.90	11	4	48	2422

4.2 Brief Summary of Further Results

Table 2 shows results for CLUSTER DELETION. The previous bound on the search tree size was $O(1.77^k)$ [5].

We have also applied our automated approach to NP-complete VERTEX DELETION problems, e.g.:

Input: A graph $G = (V, E)$, and a nonnegative integer k .

Question in the case of CLUSTER VERTEX DELETION: Can we transform G , by deleting at most k vertices, into a set of disjoint cliques?

Question in the case of TRIANGLE VERTEX DELETION: Can we transform G , by deleting at most k vertices, into a graph that contains no triangle as vertex-induced subgraph?

Each of these two graph problems specifies a forbidden vertex-induced subgraph of three vertices, i.e., an induced P_3 or an induced K_3 , respectively.

Results and Discussion. See Table 3 and Table 4. Using the enumeration without non-trivial expansion for CLUSTER VERTEX DELETION, we could only process graphs with up to eight vertices since the number of graphs to be inspected is huge. This yields the same worst-case branching number 2.27 as we have from the 3-HITTING SET algorithm in [10].² Using a cutoff value reduces

² One can prove that the VERTEX DELETION problems considered in this paper can be easily framed as instances of d -HITTING SET. See the full version of the paper.

Table 4. Results for TRIANGLE VERTEX DELETION: (1) Expansion scheme utilizing problem-specific expansion rule; (2) additionally, with cutoff

	size	time	isom	concat	graphs	maxbn	avgbn	bvlen	medlen	maxlen	bvset
(1)	8	9 min	0%	46%	7225	2.47	1.97	13	5	42	384
(2)	8	23 sec	0%	43%	433	2.47	2.10	13	4	34	355
(2)	9	10 hours	0%	56%	132370	2.42	1.97	17	5	66	1842

the number of graphs to be inspected drastically and, thus, allows us to inspect graphs with up to ten vertices. In this way, we can improve the worst-case branching number to 2.26.

When comparing the two approaches, we observe that, when using cutoff values, the average branching number (avgbn) of the computed set of branching rules becomes larger compared to the case where cutoff values were not used. The explanation is that the branching is not further improved as soon as it yields a branching number better than the cutoff value.

Finally, in Fig. 3, we compare, for different graph modification problems, the decrease of the worst-case branching numbers when increasing the size of the considered subgraphs. Only some of them have been defined in this extended abstract. In most cases, inspecting larger subgraphs yields an improved worst-case branching number.

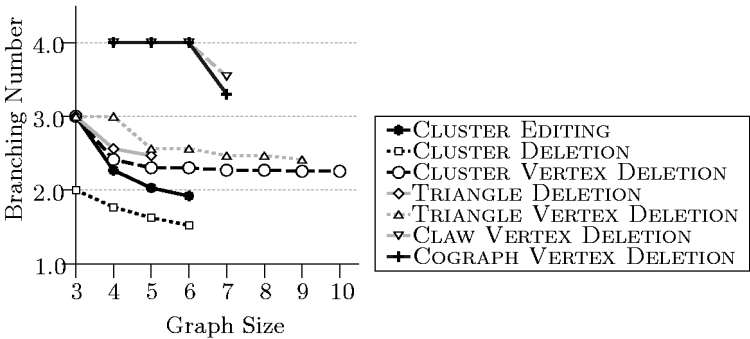


Fig. 3. Worst-case branching number depending on size of considered subgraphs

We summarize some further observations as follows: In many cases, the average branching number of the computed branching rules is significantly smaller than the worst case. For smaller graphs, a larger part of the running time is spent on the isomorphism tests. With growing graph sizes, the part of the running time spent on the administration of branching vectors in the search tree becomes larger and often takes close to 100 percent of the running time. The resulting branching rules branch, even for large graphs, only into a moderate

number of branching cases, e.g., into at most 11 branching cases in CLUSTER VERTEX DELETION when inspecting graphs of size 10.

5 Conclusion

It remains future work to extend our framework in order to directly translate the computed case distinctions into “executable search tree algorithm code” and to test the in this way implemented algorithms empirically. Our approach has two main computational bottlenecks: The enumeration of all non-isomorphic graphs up to a certain size and the concatenation of (large sets of) branching rules in our meta search tree. The approach seems to have the potential to establish new ways for proving upper bounds on the running time of NP-hard combinatorial problems; for instance, we recently succeeded in finding a non-trivial bound for the NP-hard DOMINATING SET problem with a maximum vertex degree of 3. Independently from this work, Frank Kammer and Torben Hagerup (Frankfurt/Main) informed us about ongoing related work concerning computer-generated proofs for upper bounds on NP-hard combinatorial problems.

References

1. J. Chen and I. Kanj. Improved exact algorithms for MAX-SAT. In *Proc. 5th LATIN*, number 2286 in LNCS, pp. 341–355. Springer, 2002.
2. J. Chen, I. Kanj and W. Jia. Vertex cover: further observations and further improvements. *Journal of Algorithms*, 41:280–301, 2001.
3. V. Dahllöf and P. Jonsson. An algorithm for counting maximum weighted independent sets and its applications. In *Proc. 13th ACM SODA*, pp. 292–298, 2002.
4. L. Drori and D. Peleg. Faster exact solutions for some NP-hard problems. *Theoretical Computer Science*, 287(2):473–499, 2002.
5. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: fixed-parameter algorithms for clique generation. In *Proc. 5th CIAC*, number 2653 in LNCS, pp. 108–119. Springer, 2003.
6. E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.
7. J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comp. Sys. Sci.*, 20(2):219–230, 1980.
8. B. D. McKay. **nauty** user’s guide (version 1.5). Technical report TR-CS-90-02, Australian National University, Department of Computer Science, 1990.
9. A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113:109–128, 2001.
10. R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, to appear, 2003.
11. R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for Weighted Vertex Cover. *Journal of Algorithms*, 47(2):63–77, 2003.
12. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.
13. R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. In *Proc. 28th WG*, number 2573 in LNCS, pp. 379–390. Springer, 2002.

Boolean Operations on 3D Selective Nef Complexes: Data Structure, Algorithms, and Implementation^{*}

Miguel Granados, Peter Hachenberger, Susan Hert, Lutz Kettner, Kurt Mehlhorn, and
Michael Seel

Max-Planck Institut für Informatik, Saarbrücken mgranado@cad4.eafit.edu.co,
{hachenberger|hert|kettner|mehlhorn}@mpi-sb.mpg.de michael.seel@sap.com

Abstract. We describe a data structure for three-dimensional Nef complexes, algorithms for boolean operations on them, and our implementation of data structure and algorithms. Nef polyhedra were introduced by W. Nef in his seminal 1978 book on polyhedra. They are the closure of half-spaces under boolean operations and can represent non-manifold situations, open and closed boundaries, and mixed dimensional complexes. Our focus lies on the generality of the data structure, the completeness of the algorithms, and the exactness and efficiency of the implementation. In particular, all degeneracies are handled.

1 Introduction

Partitions of three space into cells are a common theme of solid modeling and computational geometry. We restrict ourselves to partitions induced by planes. A set of planes partitions space into cells of various dimensions. Each cell may carry a label. We call such a partition together with the labelling of its cells a *selective Nef complex (SNC)*. When the labels are boolean ($\{in, out\}$) the complex describes a set, a so-called *Nef polyhedron* [23]. Nef polyhedra can be obtained from halfspaces by boolean operations union, intersection, and complement. Nef complexes slightly generalize Nef polyhedra through the use of a larger set of labels. Figure 1 shows a Nef polyhedron.

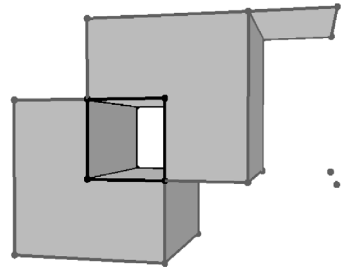


Fig. 1. A Nef polyhedron with non-manifold edges, a dangling facet, two isolated vertices, and an open boundary in the tunnel.

Nef polyhedra and complexes are quite general. They can model non-manifold solids, unbounded solids, and objects comprising parts of different dimensionality. Is this generality needed?

^{*} Work on this paper has been partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces), and by the ESPRIT IV LTR Project No. 28155 (GALIA). We thank Sven Havemann and Peter Hoffmann for helpful discussions.

1. Nef polyhedra are the smallest family of solids containing the half-spaces and being closed under boolean operations. In particular, boolean operations may generate non-manifold solids, e.g., the symmetric difference of two cubes in Figure 1, and lower dimensional features. The latter can be avoided by regularized operations.
2. In a three-dimensional earth model with different layers, reservoirs, faults, etc., one can use labels to distinguish between different soil types. Furthermore, in this application we encounter complex topology, for example, non-manifold edges.
3. In machine tooling, we may want to generate a polyhedron Q by a cutting tool M . When the tool is placed at a point p in the plane, all points in $p + M$ are removed. Observe, when the cutting tool is modeled as a closed polyhedron and moved along a path L (including its endpoints) an open polyhedron is generated. Thus open and closed polyhedra need to be modeled. The set of legal placements for M is the set $C = \{p; p + M \cap Q = \emptyset\}$; C may also contain lower dimensional features. This is one of the examples where Middleditch [22] argues that we need more than regularized boolean operations. In the context of robot motion planning this example is referred to as *tight passages*, see [14] for the case of planar configuration spaces.

SNCs can be represented by the underlying plane arrangement plus the labeling of its cells. This representation is space-inefficient if adjacent cells frequently share the same label and it is time-inefficient since navigation through the structure is difficult.

We give a more compact and unique representation of SNCs, algorithms realizing the (generalized) set operations based on this representation, and an implementation. The uniqueness of the representation, going back to Nef's work [23], is worth emphasizing; two point sets are the same if and only if they have the same representation.

The current implementation supports the construction of Nef polyhedra from manifold solids, boolean operations (union, intersection, complement, difference, symmetric difference), topological operations (interior, closure, boundary), rotations by rational rotation matrices (arbitrary rotation angles are approximated up to a specified tolerance [7]). Our implementation is exact. We follow the exact computation paradigm to guarantee correctness; floating point filtering is used for efficiency. .

Our representation and algorithm refine the results of Rossignac and O'Connor [24], Weiler [31], Gursoz, Choi, and Prinz [13], and Dobrindt, Mehlhorn, and Yvinec [10], and Fortune [12]; see Section 7 for a detailed comparison. Our structure explicitly describes the geometry around each vertex in a so-called sphere map; see Figure 4.

The paper is structured as follows: Nef polyhedra are reviewed in Section 2, our data structure is defined in Section 3, and the algorithms for generalized set operations are described in Section 4. We discuss their complexity in Section 5. We argue that our structure can be refined so as to handle special cases (almost) as efficient as the special purpose data structures. The status of the implementation is discussed in Section 6. We relate our work to previous work in Section 7 and offer a short conclusion in Section 8.

2 Theory of Nef Polyhedra

We repeat a few definitions and facts about Nef polyhedra [23] that we need for our data structure and algorithms. The definitions here are presented for arbitrary dimensions, but we restrict ourselves in the sequel to three dimensions.

Definition 1 (Nef polyhedron). A Nef-polyhedron in dimension d is a point set $P \subseteq \mathbb{R}^d$ generated from a finite number of open halfspaces by set complement and set intersection operations.

Set union, difference and symmetric difference can be reduced to intersection and complement. Set complement changes between open and closed halfspaces, thus the topological operations *boundary*, *interior*, *exterior*, *closure* and *regularization* are also in the modeling space of Nef polyhedra. In what follows, we refer to Nef polyhedra whenever we say polyhedra.

A face of a polyhedron is defined as an equivalence class of *local pyramids* that are a characterization of the local space around a point.

Definition 2 (Local pyramid). A point set $K \subseteq \mathbb{R}^d$ is called a cone with apex 0, if $K = \mathbb{R}^+ K$ (i.e., $\forall p \in K, \forall \lambda > 0 : \lambda p \in K$) and it is called a cone with apex x , $x \in \mathbb{R}^d$, if $K = x + \mathbb{R}^+(K - x)$. A cone K is called a pyramid if K is a polyhedron.

Now let $P \in \mathbb{R}^d$ be a polyhedron and $x \in \mathbb{R}^d$. There is a neighborhood $U_0(x)$ of x such that the pyramid $Q := x + \mathbb{R}^+((P \cap U(x)) - x)$ is the same for all neighborhoods $U(x) \subseteq U_0(x)$. Q is called the local pyramid of P in x and denoted $\text{Pyr}_P(x)$.

Definition 3 (Face). Let $P \in \mathbb{R}^d$ be a polyhedron and $x, y \in \mathbb{R}^d$ be two points. We define an equivalence relation $x \sim y$ iff $\text{Pyr}_P(x) = \text{Pyr}_P(y)$. The equivalence classes of $\sim$ are the faces of P . The dimension of a face s is the dimension of its affine hull, $\dim s := \dim \text{aff } s$.

In other words, a face s of P is a maximal non-empty subset of $\mathbb{R}^d$ such that all of its points have the same local pyramid Q denoted $\text{Pyr}_P(s)$. This definition of a face partitions $\mathbb{R}^d$ into faces of different dimension. A face s is either a subset of P , or disjoint from P . We use this later in our data structure and store a selection mark in each face indicating its set membership.

Faces do not have to be connected. There are only two full-dimensional faces possible, one whose local pyramid is the space $\mathbb{R}^d$ itself and the other with the empty set as a local pyramid. All lower-dimensional faces form the *boundary* of the polyhedron. As usual, we call zero-dimensional faces *vertices* and one-dimensional faces *edges*. In the case of polyhedra in space we call two-dimensional faces *facets* and the full-dimensional faces *volumes*. Faces are *relative open* sets, e.g., an edge does not contain its end-vertices.

Example 1. We illustrate the definitions with an example in the plane. Given the closed halfspaces

$$h_1 : y \geq 0, \quad h_2 : x - y \geq 0, \quad h_3 : x + y \leq 3, \quad h_4 : x - y \geq 1, \quad h_5 : x + y \leq 2,$$

we define our polyhedron $P := (h_1 \cap h_2 \cap h_3) - (h_4 \cap h_5)$. Figure 2 illustrates the polyhedron with its partially closed and partially open boundary, i.e., vertex v_4, v_5, v_6 , and edges e_4 and e_5 are not part of P . The local pyramids for the faces are $\text{Pyr}_P(f_1) = \emptyset$ and $\text{Pyr}_P(f_2) = \mathbb{R}^2$. Examples for the local pyramids of edges are the closed halfspace h_2 for the edge e_1 , $\text{Pyr}_P(e_1) = h_2$, and the open halfspace that is the complement of h_4 for the edge e_5 , $\text{Pyr}_P(e_5) = \{(x, y) | x - y < 1\}$. The edge e_3 consists actually of two disconnected parts, both with the same local pyramid $\text{Pyr}_P(e_3) = h_1$. In our data structure, we will represent the two connected components of the edge e_3 separately. Figure 3 lists all local pyramids for this example.

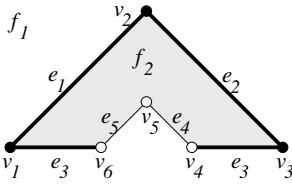


Fig. 2. Planar example of a Nef-polyhedron. The shaded region, bold edges and black nodes are part of the polyhedron, thin edges and white nodes are not.

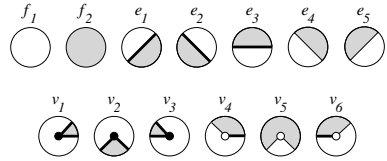


Fig. 3. Sketches of the local pyramids of the planar Nef polyhedron example. The local pyramids are indicated as shaded in the relative neighborhood in a small disc.

Definition 4 (Incidence relation). A face s is incident to a face t of a polyhedron P iff $s \subset \text{clos } t$. This defines a partial ordering $\prec$ such that $s \prec t$ iff s is incident to t .

Bieri and Nef proposed several data structures for storing Nef polyhedra in arbitrary dimensions. In the *Würzburg Structure* [6], named after the workshop location where it was first presented, all faces are stored in the form of their local pyramids, in the *Extended Würzburg Structure* the *incidences* between faces are also stored, and in the *Reduced Würzburg Structure* [5] only the local pyramids of the minimal elements in the incidence relation $\prec$ are stored. For bounded polyhedra all minimal elements are vertices. Either Würzburg structure supports Boolean operations on Nef polyhedra, neither of them does so in an efficient way. The reason is that Würzburg structures do not store enough geometry. For example, it records the faces incident to an edge, but it does not record their cyclic ordering around the edge.

3 Data Structures

In our representation for three-dimensions, we use two main structures: Sphere Maps to represent the local pyramids of each vertex and the Selective Nef Complex Representation to organize the local pyramids into a more easily accessible polyhedron representation. It is convenient (conceptually and, in particular, in the implementation) to only deal with bounded polyhedra; the reduction is described in the next section.

3.1 Bounding Nef Polyhedra. We extend infimal frames [29] already used for planar Nef polygons [28,27]. The *infimal box* is a bounding volume of size $[-R, +R]^3$ where R represents a sufficiently large value to enclose all vertices of the polyhedron. The value of R is left unspecified as an *infimal number*, i.e., a number that is finite but larger than the value of any concrete real number. In [29] it is argued that interpreting R as an infimal number instead of setting it to a large concrete number has several advantages, in particular increased efficiency and convenience.

Clipping lines and rays at this infimal box leads to points on the box that we call *frame points* or *non-standard points* (compared to the regular *standard points* inside the box). The coordinates of such points are R or $-R$ for one coordinate axis, and linear functions $f(R)$ for the other coordinates. We use linear polynomials over R as coordinate

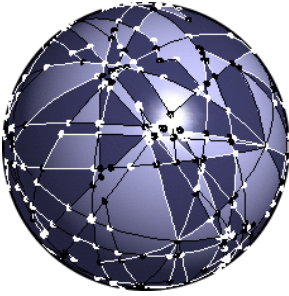


Fig. 4. An example of a sphere map. The different colors indicate selected and unselected faces.

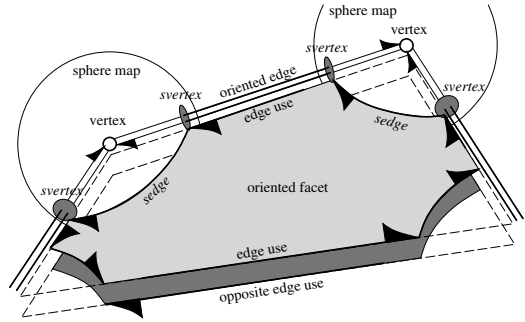


Fig. 5. An SNC. We show one facet with two vertices, their sphere maps, the connecting edges, and both oriented facets. Shells and volumes are omitted.

representation for standard points as well as for non-standard points, thus unifying the two kind of points in one representation, the *extended points*. From there we can define *extended segments* with two extended points as endpoints. Extended segments arise from clipping halfspaces or planes at the infimal box.

It is easy to compute predicates involving extended points. In fact, all predicates in our algorithms resolve to the sign evaluation of polynomial expressions in point coordinates. With the coordinates represented as polynomials in R , this leads to polynomials in R whose leading coefficient determines their signs.

We will also construct new points and segments. The coordinates of such points are defined as polynomial expressions of previously constructed coordinates. Fortunately, the coordinate polynomials stay linear even in iterated constructions.

Lemma 1. *The coordinate representation of extended points in three-dimensional Nef polyhedra is always a polynomial in R with a degree of at most one. This also holds for iterated constructions where new planes are formed from constructed (standard) intersection points. (Proof omitted due to space limitations.)*

3.2 Sphere Map. The local pyramids of each vertex are represented by conceptually intersecting the local neighborhood with a small ε -sphere. This intersection forms a planar map on the sphere (Figure 4), which together with the set-selection mark for each item forms a two-dimensional Nef polyhedron embedded in the sphere. We add the set-selection mark for the vertex and call the resulting structure the *sphere map* of the vertex. Sphere maps were introduced in [10].

We use the prefix *s* to distinguish the elements of the sphere map from the three-dimensional elements. An *svertex* corresponds to an edge intersecting the sphere. An *sedge* corresponds to a facet intersecting the sphere. Geometrically the edge forms a great arc that is part of the great circle in which the supporting plane of the facet intersects the sphere. When there is a single facet intersecting the sphere in a great circle, we get an *sloop* going around the sphere without any incident vertex. There is at most one *sloop* per vertex because a second *sloop* would intersect the first. An *sface* corresponds to a volume. This representation extends the planar Nef polyhedron representation [27].

3.3 Selective Nef Complex Representation. Having sphere maps for all vertices of our polyhedron is a sufficient but not easily accessible representation of the polyhedron. We enrich the data structure with more explicit representations of all the faces and incidences between them. We also depart slightly from the definition of faces in a Nef polyhedron; we represent the connected components of a face individually and do not implement additional bookkeeping to recover the original faces (e.g., all edges on a common supporting line with the same local pyramid) as this is not needed in our algorithms. We discuss features in the increasing order of dimension; see also Figure 5:

Edges: We store two oppositely oriented edges for each edge and have a pointer from one oriented edge to its opposite edge. Such an oriented edge can be identified with an *svertex* in a sphere map; it remains to link one *svertex* with the corresponding opposite *svertex* in the other sphere map.

Edge uses: An edge can have many incident facets (non-manifold situation). We introduce two oppositely oriented edge-uses for each incident facet; one for each orientation of the facet. An edge-use points to its corresponding oriented edge and to its oriented facet. We can identify an edge-use with an oriented *sedg*e in the sphere map, or, in the special case also with an *sloop*. Without mentioning it explicitly in the remainder, all references to *sedg*e can also refer to *sloop*.

Facets: We store oriented facets as boundary cycles of oriented edge-uses. We have a distinguished outer boundary cycle and several (or maybe none) inner boundary cycles representing holes in the facet. Boundary cycles are linked in one direction. We can access the other traversal direction when we switch to the oppositely oriented facet, i.e., by using the opposite edge-use.

Shells: The volume boundary decomposes into different connected components, the *shells*. They consist of a connected set of facets, edges, and vertices incident to this volume. Facets around an edge form a radial order that is captured in the radial order of *sedg*es around an *svertex* in the sphere map. Using this information, we can trace a shell from one entry element with a graph search. We offer this graph traversal in a visitor design pattern to the user.

Volumes: A volume is defined by a set of shells, one outer shell containing the volume and several (or maybe none) inner shells excluding voids from the volume.

For each face we store a label, e.g., a set-selection mark, which indicates whether the face is part of the solid or if it is excluded. We call the resulting data structure *Selective Nef Complex*, *SNC* for short.

4 Algorithms

Here we describe the algorithms for constructing sphere maps for a polyhedron, the corresponding SNC, and the simple algorithm that follows from these data structures for performing boolean operations on polyhedra.

4.1 Construction of a Sphere Map. We have extended the implementation of the planar Nef polyhedra in CGAL to the sphere map. We summarize the implementation of planar Nef polyhedra described in [28,27] and explain the changes needed here.

The boolean operations on the planar Nef polyhedra work in three steps—overlay, selection, and simplification—following [24]. The overlay computes the conventional planar map overlay of the two input polyhedra with a sweep-line algorithm [21, section 10.7]. In the result, each face in the overlay is a subset of a face in each input polyhedron, which we call the support of that face. The selection step computes the mark of each face in the overlay by evaluating the boolean expression on the two marks of the corresponding two supports. This can be generalized to arbitrary functions on label sets. Finally, the simplification step has to clean up the data structure and remove redundant representations.

In particular, the simplification in the plane works as follows: (i) if an edge has the same mark as its two surrounding regions the edge is removed and the two regions are merged together; (ii) if an isolated vertex has the same mark as its surrounding region the vertex is removed; (iii) and if a vertex is incident to two collinear edges and all three marks are the same then the vertex is removed and the two edges are merged. The simplification is based on Nef's theory [23,4] that provides a straightforward classification of point neighborhoods; the simplification just eliminates those neighborhoods that cannot occur in Nef polyhedra. The merge operation of regions in step (i) uses a union find data structure [8] to efficiently update the pointers in the half-edge data structure associated with the regions.

We extend the planar implementation to sphere maps in the following ways. We (conceptually) cut the sphere into two hemispheres and rotate a great arc around each hemisphere instead of a sweep line in the plane. The running time of the sphere sweep is $O((n+m+s)\log(n+m))$ for sphere maps of size n and m respectively and an output sphere map of size s . Instead of actually representing the sphere map as geometry on the sphere, we use three-dimensional vectors for the *svertices*, and three-dimensional plane equations for the support of the *sedges*. Step (iii) in the simplification algorithm needs to be extended to recognize the special case where we can get an *sloop* as result.

4.2 Classification of Local Pyramids and Simplification. In order to understand the three-dimensional boolean operations and to extend the simplification algorithm from planar Nef polyhedra to three-dimensions, it is useful to classify the topology of the *local pyramid* of a point x (the sphere map that represents the intersection of the solid with the sphere plus the mark at the center of the sphere) with respect to the dimension of a Nef face that contains x . It follows from Nef's theory [23,4] that:

- x is part of a volume iff its local sphere map is trivial (only one *sface* f^s with no boundary) and the mark f^s corresponds to the mark of x .
- x is part of a facet f iff its local sphere map consists just of an *sloop* l^s and two incident *sfaces* f_1^s, f_2^s and the mark of l^s is the same as the mark of x . And at least one of f_1^s, f_2^s has a different mark.
- x is part of an edge e iff its local sphere map consists of two antipodal *svertices* v_1^s, v_2^s that are connected by a possible empty bundle of *sedges*. The *svertices* v_1^s, v_2^s and x have the same mark. This mark is different from at least one *sedge* or *sface* in between.
- x is a vertex v iff its local sphere map is none of the above.

Of course, a valid SNC will only contain sphere maps corresponding to vertices. But some of the algorithms that follow will modify the marks and potentially invalidate this condition. We extend the simplification algorithm from planar Nef polyhedra to work directly on the SNC structure. Based on the above classification and similar to the planar case, we identify redundant faces, edges, and vertices, we delete them, and we merge their neighbors.

4.3 Synthesizing the SNC from Sphere Maps. Given the sphere maps for a particular polyhedron, we wish to form the corresponding SNC. Here we describe how this is done. The synthesis works in order of increasing dimension:

1. We identify *svertices* that we want to link together as edges. We form an encoding for each *svertex* consisting of: (a) a normalized line representation for the supporting line, e.g. the normalized Plücker coordinates of the line [30], (b) the vertex coordinates, (c) a +1 or -1 indicating whether the normalization of the line equation reversed its orientation compared to the orientation from the vertex to the *svertex*. We sort all encodings lexicographically. Consecutive pairs in the sorted sequence form an edge.
2. Edge-uses correspond to *sedges*. They form cycles around *svertices*. The cycles around two *svertices* linked as an edge have opposite orientations. Thus, corresponding *sedges* are easily matched up and we have just created all boundary cycles needed for facets.
3. We sort all boundary cycles by their normalized, oriented plane equation. We find the nesting relationship for the boundary cycles in one plane with a conventional two-dimensional sweep line algorithm.
4. Shells are found with a graph traversal. The nesting of shells is resolved with ray shooting from the lexicographically smallest vertex. Its sphere map also gives the set-selection mark for this volume by looking at the mark in the sphere map in $-x$ direction. This concludes the assembly of volumes.

4.4 Boolean Operations. We represent Nef polyhedra as SNCs. We can trivially construct an SNC for a halfspace. We can also construct it from a polyhedral surface [18] representing a closed 2-manifold by constructing sphere maps first and then synthesizing the SNC as explained in the previous section.

Based on the SNC data structure, we can implement the boolean set operations. For the set complement we reverse the set-selection mark for all vertices, edges, facets, and volumes. For the binary boolean set operations we find the sphere maps of all vertices of the resulting polyhedron and synthesize the SNC from there:

1. Find possible candidate vertices. We take as candidates the original vertices of both input polyhedra, and we create all intersection points of edge-edge and edge-face intersections. Optimizations for an early reduction of the candidate set are possible.
2. Given a candidate vertex, we find its local sphere map in each input polyhedron. If the candidate vertex is a vertex of one of the input polyhedra, its sphere map is already known. Otherwise a new sphere map is constructed on the fly. We use point location, currently based on ray shooting, to determine where the vertex lies with respect to each polyhedron.

3. Given the two sphere maps for a candidate vertex, we combine them into a resulting sphere map with boolean set operation on the surfaces of the sphere maps. The surfaces are 2D Nef polyhedra.
4. Using the simplification process described in Section 4, we determine if the resulting sphere map will be part of the representation of the result. If so, we keep it for the final SNC synthesis step.

We can also easily implement the topological operations *boundary*, *closure*, *interior*, *exterior*, and *regularization*. For example, for the boundary we deselect all volume marks and simplify the remaining SNC (Section 4). The uniqueness of the representation implies that the test for the empty set is trivial. As a consequence, we can implement for polyhedra P and Q the subset relation as $P \subset Q \equiv P - Q = \emptyset$, and the equality comparison with the symmetric difference.

5 Complexity and Optimizations

Let the total complexity of a Nef polyhedron be the number of vertices, edges, and faces. Given the sphere map representation for a polyhedron of complexity n , the synthesis of the SNC is determined by sorting the Plücker coordinates, the plane sweep for the facet cycles, and the shell classification. It runs in $O(n \log n + c \cdot T_{\uparrow})$ where $T_{\uparrow}$ is the time needed for shooting a ray to identify the nesting relationship of one of the c different shells. This is currently the cost for constructing a polyhedron from a manifold solid.

Given a polyhedron of complexity n , the complement operation runs in time linear in n . The topological operations *boundary*, *closure*, *interior*, *exterior*, and *regularization* require simplification and run in time $O(n \cdot \alpha(n))$ with $\alpha(n)$ the inverse Ackermann function from the union-find structures in the simplification algorithm.

Given one polyhedron of complexity n and another polyhedron of complexity m , the boolean set operation that produces a result of complexity k has a runtime that decomposes into three parts. First, T_I , the total time to find all edge-face and edge-edge intersections. We also subsume in T_I the time needed to locate the vertices of one polyhedron in the respective other polyhedron. Let s be the number of intersections vertices found in this step. Second, $O((n + m + s) \log(n + m))$ is the runtime for the overlay computation of all $n + m + s$ sphere map pairs. Third, after simplification of the sphere maps we are left with k maps and the SNC synthesis runtime from above applies here with the time $O(k \log k + c \cdot T_{\uparrow})$.

We have kept the runtime cost for point location and intersection separate since we argue that we can choose among different well known and efficient methods in our approach, for example, octrees [26] or binary space partition (BSP) trees [9].

The space complexity of our representation is clearly linear in our total complexity of the Nef polyhedron. However, in absolute numbers we pay for our generality in various ways. We argue to use exact arithmetic and floating point filters. However, since cascaded construction is possible, we have to store the geometry using an exact arithmetic type with unbounded precision. We further added the infimal box for unbounded polyhedra. Its coordinate representation uses a (linear) polynomial in the infimal R and thus doubles the coordinates we have to store. Both, the arithmetic and the extended kernel for the infimal box, are flexible and exchangeable based on the design principles

of CGAL. So, assuming a user can accept less general arithmetic¹ and a modeling space restricted to bounded polyhedra then we can offer already in our current implementation a choice of number type and kernel that makes the geometry part of the SNC equal to other conventional representations in size and expressiveness.

What remains is the space complexity of the connectivity description (ignoring the geometry). We compare the SNC with a typical data structure used for three-dimensional manifold meshes, the polyhedral surface in CGAL based on halfedges [18]. We need five to eight times more space for the connectivity in the SNC; five if the polyhedral surface is list based and eight if it is stored more compactly—but also less powerful—in an array. Clearly this can be a prohibitive disadvantage if the polyhedron is in most places a local manifold. Although not implemented, there is an easy optimization possible that can give the same space bounds. We can specialize the sphere maps for vertices that are locally an oriented 2-manifold to just contain a list of *svertices* and *sedges* plus two volumes. Now, assuming also that the majority of vertices has a closed boundary, we can also remove the labels from the sphere map. Whenever needed, we can reconstruct the full sphere map on the fly, or even better, we can specialize the most likely operations to work more efficiently on these specialized sphere maps to gain performance.

6 Implementation

The sphere maps and the SNC data structure with the extended kernel for the infimal box are fully implemented in CGAL² [11] with all algorithms described above. We also support the standard CGAL kernels but restricted to bounded polyhedra.

The above description breaks the algorithms down to the level of point location (for location of the candidate vertices in the input polyhedra), ray shooting (for assembling volumes in the synthesis step), and intersection finding among the geometric primitives. The current implementation uses inefficient but simple and complete implementations for these substeps. It supports the construction of Nef polyhedra from manifold solids [18], boolean operations (union, intersection, complement, difference, symmetric difference), topological operations (interior, closure, boundary, regularization), rotations by rational rotation matrices (arbitrary rotation angles are approximated up to a specified tolerance [7]). Our implementation is exact. We follow the exact computation paradigm to guarantee correctness; floating point filtering is used for efficiency.

The implementation of the sphere map data structure and its algorithms has about 9000 lines of code, and the implementation of the SNC structure with its algorithms and the visualization graphics code in OpenGL has about 15000 lines of code. Clearly, the implementation re-uses parts of CGAL; in particular the geometry, the floating point filters, and some data structures.

A bound on the necessary arithmetic precision of the geometric predicates and constructions is of interest in geometric algorithms. Of course, Nef-polyhedra can be used in cascaded constructions that lead to unbounded coordinate growth. However, we can

¹ For example, bounded depth of construction or interval arithmetic that may report that the accuracy is not sufficient for a certain operation.

² <<http://www.cgal.org>>

summarize here that the algebraic degree is less than ten in the vertex coordinates for all predicates and constructions. The computations of the highest degree are in the plane sweep algorithm on the local sphere map with predicates expressed in terms of the three-dimensional geometry.

We support the construction of a Nef polyhedron from a manifold solid defined on vertices. Nef polyhedra are also naturally defined on plane equations and combined with CGAL's flexibility one can realize schemes where coordinate growth is handled favorably with planes as defining geometry [12].

7 Comparison to Extant Work

Data structures for solids and algorithms for boolean operations on geometric models are among the fundamental problems in solid modeling, computer aided design, and computational geometry [16,20,25,15,12]. In their seminal work, Nef and, later, Bieri and Nef [23,6] developed the theory of Nef polyhedra. Dobrindt, Mehlhorn, and Yvinec [10] consider Nef polyhedra in three-space and give an $O((n+m+s)\log(n+m))$ algorithm for intersecting a general Nef polyhedron with a convex one; here n and m are the sizes of the input polyhedra and s is the size of the output. The idea of the sphere map is introduced in their paper (under the name local graph). They do not discuss implementation details. Seel [27,28] gives a detailed study of planar Nef polyhedra; his implementation is available in CGAL.

Other approaches to non-manifold geometric modeling are due to Rossignac and O'Connor [24], Weiler [31], Karasick [17], Gursoz, Choi, and Prinz [13], and Fortune [12]. Rossignac and O'Connor describe modeling by so-called *selective geometric complexes*. The underlying geometry is based on algebraic varieties. The corresponding point sets are stored in selective cellular complexes. Each cell is described by its underlying extent and a subset of cells of the complex that build its boundary. The non-manifold situations that occur are modeled via the incidence links between cells of different dimension. The incidence structure of the cellular complex is stored in a hierarchical but otherwise unordered way. No implementation details are given.

Weiler's radial-edge data structure [31] and Karasick's star-edge boundary representation are centered around the non-manifold situation at edges. Both present ideas about how to incorporate the topological knowledge of non-manifold situations at vertices; their solutions are, however, not complete. Cursoz, Choi and Prinz [13] extend the ideas of Weiler and Karasick and center the design of their non-manifold modeling structure around vertices. They introduce a cellular complex that subdivides space and that models the topological neighborhood of vertices. The topology is described by a spatial subdivision of an arbitrarily small neighborhood of the vertex. Their approach gives thereby a complete description of the topological neighborhood of a vertex.

Fortune's approach centers around plane equations and uses symbolic perturbation of the planes' distances to the origin to eliminate non-manifold situations and lower-dimensional faces. Here, a 2-manifold representation is sufficient. The perturbed polyhedron still contains the degeneracies, now in the form of zero-volume solids, zero-length edges, etc. Depending on the application, special post-processing of the polyhedron might be necessary, for example, to avoid meshing a zero-volume solid. Post-processing

was not discussed in the paper and it is not clear how expensive it would be. The direction of perturbation, i.e., towards or away from the origin, can be used to model open and closed boundaries of facets.

We improve the structure of Gursoz et al. with respect to storage requirements and provide a more concrete description with respect to the work of Dobrindt et al. as well as a first implementation. Our structure provides maximal topological information and is centered around the local view of vertices of Nef polyhedra. We detect and handle all degenerate situations explicitly, which is a must given the generality of our modeling space. The clever structure of our algorithms helps to avoid the combinatorial explosion of special case handling. We use exact arithmetic to achieve correctness and robustness, combined with floating point filters based on interval arithmetic, to achieve speed.

That we can quite naturally handle all degeneracies, including non-manifold structures, as well as unbounded objects and produce always the correct mathematical result differentiates us from other approaches. Previous approaches using exact arithmetic [1, 2, 3, 12, 19] work in a less general modeling space, some unable to handle non-manifold objects and none able to handle unbounded objects.

8 Conclusion and Future Directions

We achieved our goal of a complete, exact, and correct implementation of boolean operations on a very general class of polyhedra in space. The next step towards practicability is the implementation of faster algorithms for point location, ray shooting, intersection finding, and the specialized compact representation of sphere maps for manifold vertices. Useful extensions with applications in exact motion planning are Minkowski sums and the subdivision of the solid into simpler shapes, e.g., a trapezoidal or convex decomposition in space.

For ease of exposition, we restricted the discussion to boolean flags. Larger label sets can be treated analogously.

Nef complexes are defined by planes. We plan to extend the data structure and algorithms to complexes defined by curved surfaces [24, 15].

References

1. A. Agrawal and A. G. Requicha. A paradigm for the robust design of algorithms for geometric modeling. *Computer Graphics Forum*, 13(3):33–44, 1994.
2. R. Banerjee and J. Rossignac. Topologically exact evaluation of polyhedra defined in CSG with loose primitives. *Computer Graphics Forum*, 15(4):205–217, 1996.
3. M. Benouamer, D. Michelucci, and B. Peroche. Error-free boundary evaluation based on a lazy rational arithmetic: a detailed implementation. *Computer-Aided Design*, 26(6), 1994.
4. H. Bieri. Nef polyhedra: A brief introduct. *Comp. Suppl. Springer Verlag*, 10:43–60, 1995.
5. H. Bieri. Two basic operations for Nef polyhedra. In *CSG 96: Set-theoretic Solid Modelling: Techniques and Applications*, pages 337–356. Information Geometers, April 1996.
6. H. Bieri and W. Nef. Elementary set operations with d -dimensional polyhedra. In *Comput. Geom. and its Appl.*, LNCS 333, pages 97–112. Springer Verlag, 1988.
7. J. Canny, B. R. Donald, and E. K. Ressler. A rational rotation method for robust geometric algorithms. In *Proc. ACM Sympos. Comput. Geom.*, pages 251–260, 1992.

8. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introd. to Algorithms*. MIT Press, 1990.
9. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer Verlag, 1997.
10. K. Dobrindt, K. Mehlhorn, and M. Yvinec. A complete and efficient algorithm for the intersection of a general and a convex polyhedron. In *Proc. 3rd Workshop Alg. Data Struct.*, LNCS 709, pages 314–324, 1993.
11. A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.*, 30(11):1167–1202, 2000.
12. S.J. Fortune. Polyhedral modelling with multiprecision integer arithmetic. *Computer-Aided Design*, 29:123–133, 1997.
13. E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based representation of non-manifold boundaries. *Geometric Modeling for Product Engineering*, 23(1):107–130, 1990.
14. D. Halperin. Robust geometric computing in motion. *Int. J. of Robotics Research*, 21(3):219–232, 2002.
15. M. Hemmer, E. Schömer, and N. Wolpert. Computing a 3-dimensional cell in an arrangement of quadrics: Exactly and actually! In *ACM Symp. on Comp. Geom.*, pages 264–273, 2001.
16. C. M. Hoffmann. *Geometric and Solid Modeling – An Introd.* Morgan Kaufmann, 1989.
17. M. Karasick. *On the Representation and Manipulation of Rigid Solids*. Ph.D. thesis, Dept. Comput. Sci., McGill Univ., Montreal, PQ, 1989.
18. L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 13:65–90, 1999.
19. J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic. In *Proc. ACM Solid Modeling*, 1997.
20. M. Mäntylä. *An Introd. to Solid Modeling*. Comp. Science Press, Rockville, Maryland, 1988.
21. K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
22. A. E. Middleditch. "The bug" and beyond: A history of point-set regularization. In *CSG 94 Set-theoretic Solid Modelling: Techn. and Appl.*, pages 1–16. Inform. Geom. Ltd., 1994.
23. W. Nef. *Beiträge zur Theorie der Polyeder*. Herbert Lang, Bern, 1978.
24. J. R. Rossignac and M. A. O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wozny, J. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*. North-Holland, 1989.
25. J. R. Rossignac and A. G. Requicha. Solid modeling. <http://citeseer.nj.nec.com/209266.html>.
26. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
27. M. Seel. Implementation of planar Nef polyhedra. Research Report MPI-I-2001-1-003, MPI für Informatik, Saarbrücken, Germany, August 2001.
28. M. Seel. *Planar Nef Polyhedra and Generic Higher-dimensional Geometry*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 5. December 2001.
29. M. Seel and K. Mehlhorn. Infimaximal frames: A technique for making lines look like segments. to appear in *Comp. Geom. Theory and Appl.*, www.mpi-sb.mpg.de/~mehlhorn/ftp/InfiFrames.ps, 2000.
30. J. Stolfi. *Oriented Projective Geometry: A Framework for Geometric Computations*. Academic Press, New York, NY, 1991.
31. K. Weiler. The radial edge structure: A topological representation for non-manifold geometric boundary modeling. In M. J. Wozny, H. W. McLaughlin, and J. L. Encarnação, editors, *Geom. Model. for CAD Appl.*, pages 3–36. IFIP, May 12–16 1988.

Fleet Assignment with Connection Dependent Ground Times

Sven Grothklags

University of Paderborn, Department of Computer Science
Fürstenallee 11, 33102 Paderborn, Germany
sven@uni-paderborn.de

Abstract. Given a flight schedule, which consists of a set of flights with specified departure and arrival times, a set of aircraft types and a set of restrictions, the airline fleet assignment problem (FAP) is to determine which aircraft type should fly each flight. As the FAP is only one step in a sequence of several optimization problems, important restrictions of later steps should also be considered in the FAP. This paper shows how one type of these restrictions, connection dependent ground times, can be added to the fleet assignment problem and presents three optimization methods that can solve real-world problem instances with more than 6000 legs within minutes.

1 Introduction

For operating an airline, several optimization problems have to be solved. These include network planning, aircraft and crew scheduling. In this paper we address the fleet assignment problem (FAP) in which connection dependent ground times are taken into account. Briefly, in the FAP a flight schedule is given, consisting of a set of flights without stopover, called *legs*, with departure and arrival airport, called *stations*, and departure and arrival time for every aircraft type. An aircraft type, also called *subfleet*, has to be assigned to every leg while maximizing the profit and not exceeding the given number of aircraft per subfleet.

The FAP is only one, though important, optimization problem in the planning process of an airline and these stages have impact on each other. Therefore the main operational constraints of the following stages should also be adhered by the FAP. One operational restriction, that arises in these later stages, is considering minimum turn times of aircraft that depend on the arriving *and* departing leg. We call such turn times *connection dependent ground times*. There are many situations, where connection dependent ground times are needed. A common situation occurs at stations that have distinct terminals for domestic and international flights. If an aircraft, arriving with a domestic flight, wants to proceed with an international flight, it must be towed to a different terminal, which can last more than an hour. But if it proceeds with another domestic flight, it can stay at the terminal and the minimum ground time is much shorter.

Surprisingly, the integration of connection dependent ground times into the FAP has hardly been addressed in literature so far. In this paper we therefore present three methods, one MIP based and two Local Search based approaches, to solve the FAP

with connection dependent ground times. The MIP approach combines ideas of two well-known MIP models for the FAP to get an improved model that is able to solve real-world problem instances with connection dependent ground times in reasonable time. The Local Search approach is an extension of a heuristic solver that is part of a commercial airline tool of our industrial partner Lufthansa Systems.

The FAP is of considerable importance to airlines and has therefore attracted researchers for many years, theoretically [9,11,13] and practically [1,4,6,10,14,17]. A common approach for solving the FAP is to model the problem as a mixed integer program (MIP) like in [10]. But there also exist a number of heuristic approaches [4,8,15,16,18], mainly based on Local Search. Recently, many researchers started to extend the basic FAP by adding additional restrictions [3,5] or incorporating additional optimization problems [2,15].

The paper is organized as follows. In Section 2 the FAP with connection dependent ground times is defined as a mathematical model. Section 3 introduces a new MIP model for the FAP and in Section 4 a Local Search based heuristic is presented. In Section 5 a generalized heuristic preprocessing technique is described and Section 6 reports on the experiments made. The paper ends with concluding remarks in Section 7.

2 Problem Definition

The FAP occurs in two flavors during the planning process of an airline. In the midterm strategic planning the FAP is solved for a typical cyclic time period, typically one day or one week. During short-term tactical planning the FAP must be solved for a concrete fully-dated time interval of up to six weeks.

2.1 Input Data and Notations

For simplicity and ease of notation we restrict ourselves to the non-cyclic FAP for the rest of the paper. Nevertheless note that all presented models, algorithms and preprocessing techniques can be easily adopted for cyclic FAPs. An instance of an FAP consists of the following input data:

$\mathcal{F}$	set of available subfleets
N_f	number of aircraft available for subfleet $f \in \mathcal{F}$
$\mathcal{L}$	set of legs (schedule)
$\mathcal{F}_l$	set of subfleets that can be assigned to leg $l \in \mathcal{L}$, $\mathcal{F}_l \subseteq \mathcal{F}$
$p_{l,f}$	profit of leg $l \in \mathcal{L}$ when flown by subfleet $f \in \mathcal{F}_l$
s_l^{dep}	departure station (airport) of leg $l \in \mathcal{L}$
s_l^{arr}	arrival station of leg $l \in \mathcal{L}$
$t_{l,f}^{dep}$	departure time of leg $l \in \mathcal{L}$ when flown by subfleet $f \in \mathcal{F}_l$
$t_{l,f}^{arr}$	arrival time of leg $l \in \mathcal{L}$ when flown by subfleet $f \in \mathcal{F}_l$
$g(k, l, f)$	minimum ground time needed at station s_k^{arr} when an aircraft of subfleet $f \in \mathcal{F}$ operates legs $k \in \mathcal{L}$ and $l \in \mathcal{L}$ successively; $s_k^{arr} = s_l^{dep}$ must hold

The only difference of the FAP with connection dependent ground times compared to the classical FAP is the minimum ground time function $g(k, l, f)$, which depends on the arriving *and* departing leg. In the classical FAP the minimum ground time only depends on the arriving flight k and the subfleet f and can therefore directly be incorporated into the arrival time $t_{k,f}^{arr}$.

2.2 Connection Network

The FAP with connection dependent ground times described here can be defined by a well-known IP based on a flow network, the connection network ([1]). The IP consists of $|\mathcal{F}|$ flow networks, one for each subfleet. The legs are the nodes of the network and arcs represent possible connections between legs. We define the set of valid successors $C_{l,f}$ (predecessors $C_{l,f}^{-1}$) of a leg l when flown by subfleet f :

$$C_{l,f} = \left\{ k \in \mathcal{L} \mid f \in \mathcal{F}_k, s_l^{arr} = s_k^{dep}, t_{l,f}^{arr} + g(l, k, f) \leq t_{k,f}^{dep} \right\} \cup \{*\}$$

$$C_{l,f}^{-1} = \left\{ k \in \mathcal{L} \mid f \in \mathcal{F}_k, s_k^{arr} = s_l^{dep}, t_{k,f}^{arr} + g(k, l, f) \leq t_{l,f}^{dep} \right\} \cup \{*\}$$

$C_{l,f}$ contains all possible legs, that an aircraft of subfleet f can proceed with *after* operating leg l . Therefore a leg $k \in C_{l,f}$ must be compatible with subfleet f , must depart at the arrival station of leg l and the ground time between the arrival time of leg l and the departure time of leg k must not be less than the minimum ground time of the legs l and k . The additional element $*$ is used if the aircraft does not fly any further leg until the end of the planning period. $C_{l,f}^{-1}$ contains all possible legs, that an aircraft can fly *before* operating leg l , and $*$ means that l is the first leg of the aircraft.

The connection network IP uses binary variables $x_{k,l,f}$. $x_{k,l,f}$ is one iff legs k and l are flown successively by an aircraft of subfleet f . $x_{*,l,f}$ ($x_{l,*,f}$) is used for "connections" without predecessor (successor), that is, l is the first (last) leg flown by an aircraft of subfleet f during the planning period. With these notations we can write the FAP as follows:

$$\text{maximize} \quad \sum_{k \in \mathcal{L}} \sum_{f \in \mathcal{F}_k} p_{k,f} \left(\sum_{l \in C_{k,f}} x_{k,l,f} \right) \quad (1)$$

$$\text{subject to} \quad \sum_{f \in \mathcal{F}_k} \sum_{l \in C_{k,f}} x_{k,l,f} = 1 \quad \forall k \in \mathcal{L} \quad (2)$$

$$\sum_{k \in C_{l,f}^{-1}} x_{k,l,f} - \sum_{m \in C_{l,f}} x_{l,m,f} = 0 \quad \forall l \in \mathcal{L}, f \in \mathcal{F}_l \quad (3)$$

$$\sum_{l \in \mathcal{L}} x_{*,l,f} \leq N_f \quad \forall f \in \mathcal{F} \quad (4)$$

$$x_{k,l,f} \in \{0, 1\} \quad \forall k \in \mathcal{L}, f \in \mathcal{F}_k, l \in C_{k,f} \quad (5)$$

Condition (2) and (5) ensure that every leg is flown by exactly one subfleet. Equation (3) models the flow conservation and condition (4) limits the number of aircraft of each subfleet by limiting the *total inflow* of each subfleet. The objective function (1) maximizes the total profit.

Unfortunately, the connection network is impractical for real-world instances, because the number of binary variables grows quadratically with the number of legs. Even when using sophisticated preprocessing techniques, only small FAPs (a few hundred legs) can be solved by this model.

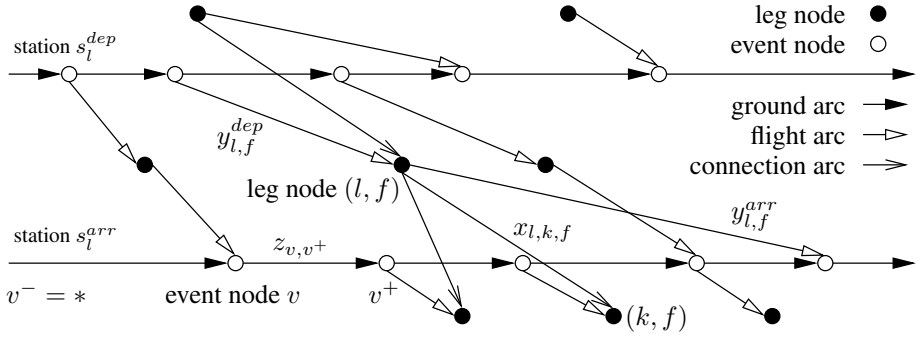


Fig. 1. Part of the flow network of subfleet f used by the new MIP model.

3 New MIP Model

In this section we present a new MIP model for the FAP with connection dependent ground times. The new model is a hybrid of the connection network introduced in Section 2.2 and the so called time space network ([10]). The time space network is probably the most popular method to solve the FAP, but it cannot handle connection dependent ground times.

The nodes in the time space network are the *flight events* on the stations (*possible arrivals and departures of aircraft*). The edges of the network are comprised of flight arcs and ground arcs. A flight arc connects a departure and an arrival event of one leg flown by a specific subfleet. The ground arcs connect two subsequent flight events on a station. A flight event can be identified by a triple (t, s, f) , where t is the time of arrival (or departure), s is the station of arrival (or departure) and f is the subfleet the event belongs to.

What makes the time space network unsuitable for dealing with connection dependent ground times is the fact, that it allows *all* connections (k, l) to be established if $t_{k,f}^{arr} \leq t_{l,f}^{dep}$ holds. The main idea for our new model is to increase $t_{k,f}^{arr}$ in such a way that *all* later departing legs form valid connections. By doing so, we may miss some valid successors of leg k , that depart earlier than the adjusted arrival time of leg k , and these connections are handled explicitly like in the connection network.

The adjusted arrival time of a leg l when flown by subfleet f can be computed by:

$$t'_{l,f} = \max \left\{ t_{k,f}^{dep} + 1 \mid f \in \mathcal{F}_k, s_l^{arr} = s_k^{dep}, t_{k,f}^{dep} < t_{l,f}^{arr} + g(l, k, f) \right\} \cup \{ t_{l,f}^{arr} \}$$

The expression simply determines the latest "compatible" departing leg, which does not form a valid connection with leg l , and sets the adjusted arrival time $t'_{l,f}$ of leg l one time unit above. If no such invalid departing leg exists, the arrival time is left unchanged. Knowing the adjusted arrival time, we can define the set of "missed" valid successors $C_{l,f}$ (predecessors $C_{l,f}^{-1}$):

$$C_{l,f} = \left\{ k \in \mathcal{L} \mid f \in \mathcal{F}_k, s_l^{arr} = s_k^{dep}, t_{l,f}^{arr} + g(l, k, f) \leq t_{k,f}^{dep}, t_{k,f}^{dep} < t'_{l,f} \right\}$$

$$C_{l,f}^{-1} = \left\{ k \in \mathcal{L} \mid f \in \mathcal{F}_k, s_k^{arr} = s_l^{dep}, t_{k,f}^{arr} + g(k, l, f) \leq t_{l,f}^{dep}, t_{l,f}^{dep} < t'_{k,f} \right\}$$

Like the connection and time space network, our new MIP model consists of $|\mathcal{F}|$ flow networks. Each flow network consists of two different kind of nodes: leg nodes, that correspond to the nodes of the connection network, and event nodes, that correspond to the nodes of the time space network. The edges of the network are comprised of the flight and ground arcs of the time space network and the connection arcs of the connection network. Like in the time space network, ground arcs connect successive event nodes of a station. The flight arcs connect the departure and arrival event of a leg with its leg node. Finally, the connection arcs, that run between leg nodes, establish the "missed" valid connections. See Figure 1 for an example.

We need the following notations to define the new MIP model:

$\mathcal{V}$ set of all flight events

$v_{l,f}^{dep} = (t_{l,f}^{dep}, s_l^{dep}, f)$; flight event that corresponds to the departure of leg l when flown by subfleet f

$v_{l,f}^{arr} = (t'_{l,f}, s_l^{arr}, f)$; flight event that corresponds to the arrival of leg l when flown by subfleet f

v^+ subsequent flight event of $v \in \mathcal{V}$ on the same station of the same subfleet; * if v is the last flight event

v^- preceding flight event of $v \in \mathcal{V}$ on the same station of the same subfleet; * if v is the first flight event

$\mathcal{V}_f^*$ set of flight events of subfleet f that have no predecessor

The new MIP model uses binary variables $x_{k,l,f}$, $y_{l,f}^{dep}$ and $y_{l,f}^{arr}$. $x_{k,l,f}$ directly corresponds to the x -variables in the connection network. $y_{l,f}^{dep}$ ($y_{l,f}^{arr}$) is the flight arc that connects the departure event $v_{l,f}^{dep}$ (arrival event $v_{l,f}^{arr}$) with the corresponding leg node. Finally, we use non-negative variables z_{v,v^+} to represent the flow on the ground arcs. With these notations the new MIP model can be written as follows:

$$\text{maximize} \quad \sum_{k \in \mathcal{L}} \sum_{f \in \mathcal{F}_k} p_{k,f} \left(y_{k,f}^{arr} + \sum_{l \in C_{k,f}} x_{k,l,f} \right) \quad (6)$$

$$\text{subject to} \quad \sum_{f \in \mathcal{F}_k} \left(y_{k,f}^{arr} + \sum_{l \in C_{k,f}} x_{k,l,f} \right) = 1 \quad \forall k \in \mathcal{L} \quad (7)$$

$$y_{l,f}^{dep} + \sum_{k \in C_{l,f}^{-1}} x_{k,l,f} - y_{l,f}^{arr} - \sum_{m \in C_{l,f}} x_{l,m,f} = 0 \quad \forall l \in \mathcal{L}, f \in \mathcal{F}_l \quad (8)$$

$$\sum_{v_{l,f}^{arr}=v} y_{l,f}^{arr} - \sum_{v_{l,f}^{dep}=v} y_{l,f}^{dep} + z_{v,v^-} - z_{v,v^+} = 0 \quad \forall v \in \mathcal{V} \quad (9)$$

$$\sum_{v \in \mathcal{V}_f^*} z_{*,v} \leq N_f \quad \forall f \in \mathcal{F} \quad (10)$$

$$x_{k,l,f} \in \{0, 1\} \quad \forall k \in \mathcal{L}, f \in \mathcal{F}_k, l \in C_{k,f} \quad (11)$$

$$y_{l,f}^{dep}, y_{l,f}^{arr} \in \{0, 1\} \quad \forall l \in \mathcal{L}, f \in \mathcal{F}_l \quad (12)$$

$$z_{v,v^+} \geq 0 \quad \forall v \in \mathcal{V} \quad (13)$$

$$z_{*,v} \geq 0 \quad \forall v \in \mathcal{V}^* \quad (14)$$

Whether a leg k is flown by subfleet f or not, can be identified by the outflow of the corresponding leg node, that is $y_{k,f}^{arr} + \sum_{l \in C_{k,f}} x_{k,l,f}$. Therefore the objective function (6) maximizes the total profit. Equation (7) ensures that every leg is operated by exactly one subfleet. Equation (8) ensures the flow conservation condition for leg nodes and equation (9) for event nodes. Condition (10) limits the number of used aircraft per subfleet by limiting the total inflow for each subfleet. Conditions (11)–(14) define the domains of the used variables.

The number of "missed" valid connections (and therefore x -variables) is quite low in practice and there are also many legs, for which $C_{l,f}$ (or $C_{l,f}^{-1}$) is empty. For these legs we can substitute $y_{l,f}^{arr}$ for $y_{l,f}^{dep} + \sum_{k \in C_{l,f}^{-1}} x_{k,l,f}$ ($y_{l,f}^{dep}$ for $y_{l,f}^{arr} + \sum_{m \in C_{l,f}} x_{l,m,f}$) and remove the corresponding leg node equation from the model. So normally, we will end up with a model that is not much larger than a corresponding classical time space network MIP. The model will collapse into a classical time space network if all sets $C_{l,f}$ and $C_{l,f}^{-1}$ are empty, which is the case for FAPs *without* connection dependent ground times.

4 Local Search Heuristics

We enhanced our previously developed Local Search based heuristics ([8]) to be able to deal with connection dependent ground times. The specialized neighborhood can be used in a Hill Climbing or simulated annealing framework to produce high quality solutions in short times. The Simulated Annealing heuristic uses an adaptive cooling schedule described in [12].

Since flight schedules evolve over time, solving the FAP does not have to start from scratch. Changes to a flight schedule are integrated into the old schedule by airline experts, so that for the real-world fleet assignment instances an initial solution is given which can then be used by our Local Search algorithms.

So clearly, the most challenging part for using Local Search to solve the FAP is how to define the neighborhood. Therefore we will restrict to this topic for the rest of the section. We allow two transitions which we call *change* and *swap*. In a nutshell, change and swap look for leg sequences, that do not use more aircraft than the current solution when moved to a different subfleet.

We initially developed our neighborhood for the FAP without considering connection dependent ground times. The following two sections shortly present the basic ideas of the original neighborhood. In Section 4.3 we describe the necessary extensions to be able to deal with connection dependent ground times.

4.1 Waiting Function

Crucial for the efficient computation of our neighborhood transitions is the *waiting function* ([7]). Given a solution to the FAP, the waiting function $W_{s,f}(t)$ counts the number of aircraft of subfleet f available on station s at time t , such that $\forall t : W_{s,f}(t) \geq 0$ and $\exists t' : W_{s,f}(t') = 0$. An island of $W_{s,f}(t)$ is an interval (t_1, t_2) , where $W_{s,f}(t)$ is strictly positive $\forall t \in (t_1, t_2)$ and $W_{s,f}(t_1) = 0 = W_{s,f}(t_2)$. At the beginning (end) of

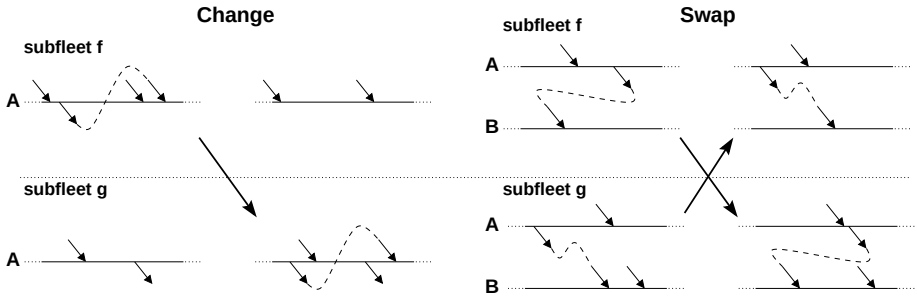


Fig. 2. Neighborhood operations: change and swap. Arcs represent incoming and outgoing legs on a station.

the planning interval $W_{s,f}(0)$ ($W_{s,f}(T)$) does not need to be zero. Every flight event (departure or arrival) of a given solution belongs exactly to one of these islands.

The waiting functions can be used in a number of ways. The value $W_{s,f}(0)$ at the beginning of the planning interval tells us the number of aircraft of subfleet f , that must be available at station s at the beginning of the planning interval to be able to operate the schedule. These values are used to determine the number of aircraft needed by a solution. Furthermore it is known, that all connections between arriving and departing legs must lie *within* the islands of the waiting function, and, more importantly, that it is always *possible* to build these connections within the islands. And finally, the waiting function is an important tool to efficiently construct leg sequences for our change and swap transitions, that do not increase the number of aircraft used.

4.2 Change and Swap

We call $(l_0, \dots, l_n)$ a *leg sequence*, if the arrival airport of l_i is the departure airport of leg l_{i+1} for all $i = 0, \dots, n-1$ and all legs $l_0, \dots, l_n$ are assigned to the same subfleet. Leg sequences are generated by a depth first search with limited degree at the search nodes. The possible successors in a leg sequence are not allowed to increase the number of aircraft used, when the leg sequence is changed or swapped to another subfleet. For the computation of a successor, the islands of the waiting function are used. The exact generation process of leg sequences is quite complex. In this paper we can only give a sketch of the two transitions, which define the neighborhood and use leg sequences as building blocks.

The Change. The change transition alters the assigned subfleet for a leg sequence which starts and ends at the same airport A . On the input of a randomly chosen leg l , with f being the subfleet currently assigned to l , and a subfleet g , the change generates a leg sequence S_f that starts with leg l and to which subfleet g can be assigned without using more aircraft than the current solution. For this, during the whole time of S_f subfleet g has to provide an aircraft waiting on the ground on A . Therefore l and the last leg of S_f must lie within one island of $W_{A,g}$.

The Swap. The swap transition exchanges the assigned subfleets f and g among two leg sequences S_f and S_g . To maintain the current distribution of aircraft at the beginning

and end of the planning interval, the two leg sequences have to start at the same airport A and end at the same airport B . Let f be the subfleet currently assigned to a randomly chosen leg l . Then, on input of leg l and a subfleet g , the swap transition computes first a leg sequence S'_f starting with l , whose legs are currently assigned to subfleet f and that can also be flown by subfleet g . The depth first search construction of leg sequence S_g also starts at station A and searches for a compatible leg sequence of legs flown by subfleet g , that ends at the same station as a *subsequence* S'_f of S_f . Then S_f and S_g form a valid swap transition.

4.3 Extensions for Connection Dependent Ground Times

Our original neighborhood for the FAP has the same difficulties with connection dependent ground times as the time space network. By using the waiting function, we can assure that a solution of the FAP (without connection dependent ground times) does not exceed the available number of aircraft, but we do not know the concrete connections between arriving and departing legs. We only know, that there *exists* at least one valid set of connections and that these connections must lie within the islands of the waiting function. But this is only true, if *all* legs, that depart later than an arriving leg l within an island, are valid successors of l , and this does not need to hold if we have to consider connection dependent ground times.

Because the waiting function has proven to be a fast and powerful tool to construct transitions for the FAP, we adhered to this concept for the FAP with connection dependent ground times and extended it by additionally storing a *valid* successor for each leg. The generation procedure of leg sequences still uses the waiting function as major tool, but it is modified to ensure, that there still is a valid successor for each leg after a transition.

The computation of successors can be solved independently for each station s and subfleet f . This can be done by calculating a complete matching on the bipartite graph of arriving and departing legs of subfleet f on station s . The nodes of this bipartite graph are the arriving and departing legs and an arriving leg k is connected with a departing leg l iff they can form a valid connection: $t_{k,f}^{arr} + g(k, l, f) \leq t_{l,f}^{dep}$. The waiting function further helps to reduce the size of the bipartite graphs, for which matchings have to be computed. As all connections must lie within islands, it is sufficient to calculate the matchings for each island separately.

The matchings are calculated by a well-known maximum-flow algorithm for the maximum matching problem for bipartite graphs. Note that these matchings only have to be computed from scratch once, namely at the beginning of the Local Search algorithm. Afterwards, we only have to deal with small updates to the islands, when single legs are added or removed. These updates happen obviously when we *switch* to a neighbor of the current solution but they also occur during the construction of leg sequences to *test* whether or not a leg sequence can be moved to another subfleet without destroying the complete matchings. In both cases, we only need to perform very few (one or two) augmentation steps of the maximum-flow algorithm to restore the complete matching or to discover that no complete matching exists anymore.

5 Preprocessing

Hane et al. ([10]) introduced a heuristic preprocessing technique, that can reduce the number of legs of an FAP instance and eliminate some of the ground arcs of the time space network. Here we present a generalization of this preprocessing technique and show how it can be applied to our new MIP model and Local Search heuristics.

Airline experts prefer schedules that use as few aircraft as possible on *small* stations. Spare aircraft should wait at large stations (hubs) where they can be used more easily in the case of schedule disruptions. A lower bound on the number of aircraft needed at a station can be determined by looking at a schedule, when it is operated by *one* artificial subfleet f^* . The subfleet f^* can fly every leg l "faster" than any real subfleet in $\mathcal{F}$, e.g. $[t_{l,f^*}^{dep}, t_{l,f^*}^{arr}] = \bigcap_{f \in \mathcal{F}_l} [t_{l,f}^{dep}, t_{l,f}^{arr}]$. The value of the waiting function $W_{s,f^*}(0)$ at the beginning of the planning period gives us the minimal number of aircraft needed at station s and, more importantly, the islands of W_{s,f^*} define intervals, in which *all* connections of a real schedule must lie if it wants to use as few aircraft at station s as the artificial subfleet f^* .

Hane et al. suggested two methods, how to reduce the size of a time space network. First of all, two legs k and l should be combined if they are the only legs in an island of W_{s,f^*} . If we want to respect the island structure of W_{s,f^*} for our schedules, k and l must always be flown by one aircraft successively and so they can be joined reducing the number of legs by one. Secondly, we can delete all ground arcs for all subfleets in the time space network that correspond to ground arcs that would run *between* islands of W_{s,f^*} . The waiting function tells us, that the flow of these ground arcs must be zero in order to achieve the minimal number of aircraft, so they can be deleted.

We propose a third reduction method that allows us to join legs that belong to islands with more than two legs. It is a generalization of the first method described above. For each island of W_{s,f^*} we compute a complete matching between arriving and departing legs. Two legs k and l are connected by an arc, if there exists at least one *real* subfleet $f \in \mathcal{F}$ that can fly k and l successively. Then we test each arc of the complete matching, if it is part of *every* possible complete matching by removing it from the bipartite graph and trying to find an alternative complete matching. If an arc is contained in every complete matching, its corresponding legs can be combined. This preprocessing step can be done in $O(n^3)$, where n is the number of flight events of the island.

The idea of joining legs can be applied to any solution method for the FAP. The idea of deleting ground arcs can also be naturally applied to our new MIP model that deals with connection dependent ground times. Besides deleting ground arcs, we can additionally remove all connection arcs that cross islands of W_{s,f^*} .

The preprocessing methods described here are heuristic ones, as they restrict the solution space and therefore may cause a loss of profit. But as they enforce a desired feature on an FAP solution and the loss of profit generally is quite low, it rather adds to the non-monetary quality of the produced assignments.

6 Experimental Evaluation

We investigated the performance (running time and solution quality) of our algorithms on several sets of real-world data. The instances are problems of recent summer and

Table 1. Properties of the problem instances tested and runtime and solution quality of Hill Climbing (HC), Simulated Annealing (SA) and MIP model with full preprocessing.

FAP	instance properties				HC		SA		MIP		
	legs	subfleets	stations	altern.	time	quality	time	quality	total time	root time	quality
A	6287	8	96	2.9	16	97.99%	302	98.81%	752	54	99.90%
B	5243	23	76	1.7	1	99.13%	13	99.84%	4	3	optimal
C	5306	21	76	4.8	5	97.82%	84	99.40%	314	125	99.96%
D	5186	20	71	4.6	5	98.12%	69	99.51%	229	124	99.98%

winter schedules from major airlines provided to us by our industrial partner Lufthansa Systems. Due to confidentiality, only percentage results on the objective values (profit) can be shown. Our experiments include a comparison of the effects of the preprocessing techniques presented in Section 5 and a comparison between the performance of FAPs with and without connection dependent ground times.

Table 1 lists the sizes of the four problem instances used in this evaluation. The column "altern." contains the average number of different subfleets a leg can be assigned to. Instance B is special because its alternative-value is very low. Many of the legs are fixed to one subfleet. This instance emerged from a scenario where only the legs arriving or departing at one specific station should be optimized.

All tests were executed on a PC workstation, Pentium III 933 MHz, with 512 MByte RAM running under RedHat Linux 7.3. We used two randomized heuristics using the neighborhood presented in Section 4, a Hill Climbing (HC) and a Simulated Annealing (SA) algorithm, implemented in C. The HC and SA results show the average value of 5 runs. Our new MIP model was solved by the branch and bound IP-solver of CPLEX 7.5. The root node was computed using the interior-point barrier-method and the sub nodes were processed by the dual simplex algorithm. The IP-solver was used in a heuristic configuration, terminating as soon as the first valid (integral) solution was found.¹ Therefore also for the MIP approach a solution quality is given which corresponds to the IP-gap reported by CPLEX. The solution quality of the Local Search heuristics is calculated relatively to the upper bound generated by the corresponding MIP run. For the CPLEX runs both, the root relaxation time and the total run time, are given. All run times are given in seconds.

Table 1 shows the run times and solution qualities of our algorithms on the benchmark set. All preprocessing techniques described in Section 5 were applied. HC is fastest, but has the worst solution quality. MIP computes near optimal solutions in reasonable time, but normally is the slowest approach. Only on instance B it is able to outperform SA due to the very special structure of instance B mentioned above. Finally, SA is a compromise between speed and quality. In general, it is not easy to decide, if a solution quality of 99% is sufficiently good. On the one hand, we are dealing with huge numbers and 1% can be hundred thousands of Euro. On the other hand, we are only given *estimations* of the profit that can be off by more than 10%, especially during strategic planning.

In Table 2 we present the effects of the preprocessing techniques of Section 5. It contains results of four different preprocessing settings that we applied to (the repre-

¹ The computation of an *optimal* solution can last days despite the small IP-gaps.

Table 2. The influence of preprocessing on our FAP algorithms.

Prepro	profit loss	legs	conn. arcs	ground arcs	flight arcs	leg equ.	rows	columns	MIP time	root time	SA time
no	–	6287	8050	12054	18168	94	14544	34349	5153	179	383
J	0.39%	5102	6143	8545	14660	731	11238	26181	910	77	336
J*	0.56%	4549	6005	7163	13028	802	9693	23355	1001	59	302
GJ*	0.61%	4549	5984	6386	12953	802	9545	22611	752	54	302

Table 3. Comparison between FAPs with and without connection dependent ground times (CDGT).

Prepro	Instance A with CDGT						Instance A without CDGT					
	rows	columns	MIP time	root time	SA time		rows	columns	MIP time	root time	SA time	
no	14544	34349	5153	179	383		14155	25930	3199	114	123	
J	11238	26181	910	77	336		10100	18915	415	47	103	
J*	9693	23355	1001	59	302		8480	16145	336	31	97	
GJ*	9545	22611	752	54	302		8313	15403	360	27	95	

sentative) instance A. Preprocessing setting "no" stands for no preprocessing, "J" joins legs of islands that only consist of two legs, "J*" additionally joins legs in bigger islands and "GJ*" uses all techniques described in Section 5. As mentioned earlier, these preprocessing techniques are heuristic ones and therefore the first column lists the loss in profit due to the preprocessing. The following columns show the effect of preprocessing on the LP-size of our MIP model. The number of legs, number of connection, ground and flight arcs, the number of leg node equations and the total size of the LP are given. The last columns contain the run times of our MIP and SA approach. As can be seen, preprocessing is crucial for the MIP approach, as it is able to significantly reduce the run times. SA does not take that much of an advantage from preprocessing.

In Table 3 we compare the run times of our FAP solvers on instances with and without connection dependent ground times to see the impact of adding them to the FAP. We took instance A and replaced its connection dependent ground times $g(k, l, f)$ by classical leg dependent ground times $g(k, f) = \min_{l \in \mathcal{L}} g(k, l, f)$. Thereby we built an instance without connection dependent ground times, similar in size and structure to instance A. Test runs were executed using the four different preprocessing settings from above. Note that our MIP approach transforms into a regular time space network for FAPs without connection dependent ground times. The table shows that the solution times increase only by a factor of 2 to 3 by introducing connection dependent ground times, both for the MIP and SA approach.

7 Conclusions

The main contribution of this paper shows, how an important operational restriction, connection dependent ground times, can be incorporated into the fleet assignment problem and that this extended FAP can be solved for real-world problem instances. We presented and evaluated three different optimization methods, two Local Search based (HC and

SA) and one MIP based approach, that have different characteristics concerning run time and solution quality. HC is fast but produces low quality solutions, MIP calculates near optimal solutions but needs longer and SA is a compromise between speed and quality.

We are currently trying to incorporate additional restrictions and extensions into the FAP. Now that we have developed a framework that can explicitly model connections for a limited time horizon after the arrival of a leg, it should be possible to include the through-assignment problem, which is part of the aircraft rotation building optimization step. Furthermore, certain punctuality restrictions, like forbidding successive connections without time buffers, can be integrated.

References

1. J. Abara. Applying integer linear programming to the fleet assignment problem. *Interfaces*, 19(4):20–28, 1989.
2. Cynthia Barnhart, Natasha L. Boland, Lloyd W. Clarke, and Rajesh G. Shenoi. Flight strings models for aircraft fleet and routing. Technical report, MIT Cambridge, 1997.
3. N. Belanger, G. Desaulniers, F. Soumis, J. Desrosiers, and J. Lavigne. Airline fleet assignment with homogeneity. Technical report, GERAD, Montréal, 2002.
4. M.A. Berge and C.A. Hopperstad. Demand driven dispatch: A method for dynamic aircraft capacity assignment, models and algorithms. *Operations Research*, 41(1):153–168, 1993.
5. L. W. Clarke, C.A. Hane, E.L. Johnson, and G.L. Nemhauser. Maintenance and crew considerations in fleet assignment. Technical report, 1994.
6. G. Desaulniers, J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Daily aircraft routing and scheduling. *Management Science*, 43(6):841–855, 1997.
7. I. Gertsbach and Yu. Gurevich. Constructing an optimal fleet for a transportation schedule. *Transportation Science*, 11(1):20–36, 1977.
8. S. Götz, S. Grothklags, G. Kliewer, and S. Tschöke. Solving the weekly fleet assignment problem for large airlines. In *MIC'99*, pages 241 – 246, 1999.
9. Z. Gu, E.L. Johnson, G.L. Nemhauser, and Y. Wang. Some properties of the fleet assignment problem. *Operations Research Letters*, 15:59–71, 1994.
10. C.A. Hane, C. Barnhart, E.L. Johnson, R.E. Marsten, G.L. Nemhauser, and G. Sigismondi. The fleet assignment problem: solving a large-scale integer program. *Mathematical Programming*, 70:211–232, 1995.
11. Timothy S. Kniker and Cynthia Barnhart. Shortcomings of the conventional fleet assignment model. Technical report, MIT Cambridge, 1998.
12. I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.
13. Ulf-Dietmar Radicke. *Algorithmen für das Fleet Assignment von Flugplänen*. Verlag Shaker, 1994.
14. R.A. Rushmeier and S.A. Kontogiorgis. Advances in the optimization of airline fleet assignment. *Transportation Science*, 31(2):159–169, 1997.
15. D. Sharma, R. K. Ahuja, and J. B. Orlin. Neighborhood search algorithms for the combined through-fleet assignment model. Talk at ISMP'00, 2000.
16. D. Sosnowska. Optimization of a simplified fleet assignment problem with metaheuristics: Simulated annealing and GRASP. In P. M. Pardalos, editor, *Approximation and Complexity in Numerical Optimization*. Kluwer Academic Publisher, 2000.
17. R. Subramanian, R.P. Sheff, J.D. Quillinan, D.S. Wiper, and R.E. Marsten. Coldstart: Fleet assignment at Delta Air Lines. *Interfaces*, 24(1):104–120, 1994.
18. K. T. Talluri. Swapping applications in a daily airline fleet assignment. *Transportation Science*, 30(3):237–248, 1996.

A Practical Minimum Spanning Tree Algorithm Using the Cycle Property^{*}

Irit Katriel¹, Peter Sanders¹, and Jesper Larsson Träff²

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany
`{irit,sanders}@mpi-sb.mpg.de`

² C&C Research Laboratories, NEC Europe Ltd., Sankt Augustin, Germany
`traff@ccrl-nece.de`

Abstract. We present a simple new (randomized) algorithm for computing minimum spanning trees that is more than two times faster than the best previously known algorithms (for dense, “difficult” inputs). It is of conceptual interest that the algorithm uses the property that the heaviest edge in a cycle can be discarded. Previously this has only been exploited in asymptotically optimal algorithms that are considered impractical. An additional advantage is that the algorithm can greatly profit from pipelined memory access. Hence, an implementation on a vector machine is up to 10 times faster than previous algorithms. We outline additional refinements for MSTs of implicitly defined graphs and the use of the central data structure for querying the heaviest edge between two nodes in the MST. The latter result is also interesting for sparse graphs.

1 Introduction

Given an undirected connected graph G with n nodes, m edges and (nonnegative) edge weights, the minimum spanning tree (MST) problem asks for a minimum total weight subset of the edges that forms a spanning tree of G .

The current state of the art in MST algorithms shows a gap between theory and practice. The algorithms used in practice are among the oldest network algorithms [2,5,10,13] and are all based on the *cut property*: a *lightest* edge leaving a set of nodes can be used for an MST. More specifically, Kruskal’s algorithm [10] is best for sparse graphs. Its running time is asymptotically dominated by the time for sorting the edges by weight. For dense graphs ($m \gg n$), the Jarník-Prim (JP) algorithm is better [5,15]. Using Fibonacci heap priority queues, its execution time is $\mathcal{O}(n \log n + m)$. Using pairing heaps [3] Moret and Shapiro [12] get quite favorable results in practice at the price of worse performance guarantees.

On the theoretical side there is a randomized linear time algorithm [6] and an almost linear time deterministic algorithm [14]. But these algorithms are usually considered impractical because they are complicated and because the constant factors in the execution time look unfavorable. These algorithms complement the cut property with the *cycle property*: a *heaviest edge* in any cycle is not needed for an MST.

^{*} Partially supported by DFG grant SA 933/1-1.

In this paper we partially close this gap. We develop a simple $\mathcal{O}(n \log n + m)$ expected time algorithm using the cycle property that is very fast on dense graphs. Our experiments show that it is more than two times faster than the JP algorithm for large dense graphs that require a large number of priority queue updates for JP. For future architectures it promises even larger speedups because it profits from pipelining for hiding memory access latency. An implementation on a vector machine shows a speedup by a factor of 10 for large dense graphs.

Our algorithm is a simplification of the linear time randomized algorithms. Its asymptotic complexity is $\mathcal{O}(m + n \log n)$. When $m \gg n \log n$ we get a linear time algorithm with small constant factors. The key component of these algorithms works as follows. Generate a smaller graph G' by selecting a random sample of the edges of G . Find a minimum spanning forest T' of G' . Then, *filter* each edge $e \in E$ using the cycle property: Discard e if it is the heaviest edge on a cycle in $T' \cup \{e\}$. Finally, find the MST of the graph that contains the edges T' and the edges that were not filtered out. Since MST edges were not discarded, this is also the MST of G .

Klein and Tarjan [8] prove that if the sample graph G' is obtained by including each edge of G independently with probability p , then the expected number of edges that are not filtered out is bounded from above by n/p . By setting $p = \sqrt{n/m}$ both recursively solved MST instances can be made small. It remains to find an efficient way to implement filtering.

King [7] suggests a filtering scheme which requires an $\mathcal{O}(n \log \frac{m+n}{n})$ preprocessing stage, after which the filtering can be done with $\mathcal{O}(1)$ time per edge (for a total of $\mathcal{O}(m)$). The preprocessing stage runs Boruvka's [2,13] algorithm on the spanning tree T' and uses the intermediate results to construct a tree B that has the vertices of G as leaves such that: (1) the heaviest edge on the path between two leaves in B is the same as the heaviest edge between them in T' . (2) B is a *full branching tree*; that is, all the leaves of B are at the same level and each internal node has at least two sons. (3) B has at most $2n$ nodes. It is then possible to apply to B Komlós's algorithm [9] for maximum edge weight queries on a full branching tree. This algorithm builds a data structure of size $\mathcal{O}(n \log(\frac{m+n}{n}))$ which can be used to find the maximum edge weight on the path between leaves u and v , denoted $F(u, v)$, in constant time. A path between two leaves is divided at their least common ancestor (LCA) into two half paths and the maximum weight on each half path is precomputed. In addition, during the preprocessing stage the algorithm generates information such that the LCA of two leaves can be found in constant time.

In Section 2 we develop a simpler filtering scheme that is based on the order in which the JP algorithm adds nodes to the MST of the sample graph G' . We show that using this ordering, computing $F(u, v)$ reduces to a single interval maximum query. This is significantly simpler to implement than Komlós's algorithm because (1) we do not need to convert T' into a different tree. (2) interval maximum computation is more structured than path maximum in a full branching tree, where nodes may have different degrees. As a consequence, the

preprocessing stage involves computation of simpler functions and needs simpler data structures.

Interval maxima can be found in constant time by applying a standard technique that uses precomputed tables of total size $\mathcal{O}(n \log n)$. The tables store prefix minima and suffix maxima [4]. We explain how to arrange these tables in such a way that $F(u, v)$ can be found using two table lookups for finding the JP-order, one exclusive-or operation, one operation finding the most significant nonzero bit, two table lookups in fused prefix and suffix tables and some shifts and adds for index calculations. These operations can be executed independently for all edges, in contrast to the priority queue accesses of the JP algorithm that have to be executed sequentially to preserve correctness.

In Section 3 we report measurements on current high-end microprocessors that show speedup up to a factor 3.35 compared to a highly tuned implementation of the JP algorithm. An implementation on a vector computer results in even higher speedup of up to 10.

2 The I-Max-Filter Algorithm

In Section 2.1 we explain how finding the heaviest edge between two nodes in an MST can be reduced to finding an interval maximum. The array used is the edge weights of the MST stored in the order in which the edges are added by the JP algorithm. Then in Section 2.2 we explain how this interval maximum can be computed using one further table lookup per node, an exclusive-or operation and a computation of the position of the most significant one-bit in an integer. In Section 2.3 we use these components to assemble the I-Max-Filter algorithm for computing MSTs.

2.1 Reduction to Interval Maxima

The following lemma shows that by renumbering nodes according to the order in which they are added to the MST by the JP algorithm, heaviest edge queries can be reduced to simple interval maximum queries.

Lemma 1. *Consider an MST $T = (\{0, \dots, n-1\}, E_T)$ where the JP algorithm (JP) adds the nodes to the tree in the order $0, \dots, n-1$. Let e_i , $0 < i < n$ denote the edge used to add node i to the tree by the JP algorithm. Let w_i , denote the weight of e_i . Then, for all nodes $u < v$, the heaviest edge on the path from u to v in T has weight $\max_{u < j \leq v} w_j$.*

Proof. By induction over v . The claim is trivially true for $v = 1$. For the induction step we assume that the claim is true for all pairs of nodes (u, v') with $u < v' < v$ and show that it is also true for the pair (u, v) . First note that e_v is on the path from u to v because in the JP algorithm u is inserted before v and v is an isolated node until e_v is added to the tree. Let $v' < v$ denote the node at the other end of edge e_v . Edge e_v is heavier than all the edges $e_{v'+1}, \dots, e_{v-1}$

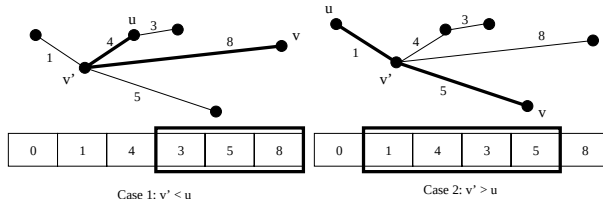


Fig. 1. Illustration of the two cases of Lemma 1. The JP algorithm adds the nodes from left to right.

because otherwise the JP algorithm would have added v , using e_v , earlier. There are two cases to consider (see Figure 1).

Case $v' \leq u$: By the induction hypothesis, the heaviest edge on the path from v' to u is $\max_{v' < j \leq u} w_j$. Since all these edges are lighter than e_v , the maximum over $w_u, \dots, w_v$ finds the correct answer w_v .

Case $v' > u$: By the induction hypothesis, the heaviest edge on the path between u and v' has weight $\max_{u < j \leq v'} w_j$. Hence, the heaviest edge we are looking for has weight $\max\{w_v, \max_{u < j \leq v'} w_j\}$. Maximizing over the larger set $\max_{u < j \leq v} w_j$ will return the right answer since e_v is heavier than the edges $e_{v'+1}, \dots, e_{v-1}$.

Lemma 1 also holds when we have the MSF of an unconnected graph rather than the MST of a connected graph. When JP spans a connected component, it selects an arbitrary node i and adds it to the MSF with $w_i = \infty$. Then the interval maximum for two nodes that are in two different components is ∞ , as it should be.

2.2 Computation of Interval Maxima

Given an array $a[0] \dots a[n-1]$, we explain how $\max a[i..j]$ can be computed in constant time using preprocessing time and space $\mathcal{O}(n \log n)$. The emphasis is on very simple and fast queries since we are looking at applications where many more than $n \log n$ queries are made. To this end we develop an efficient implementation of a basic method described in [4, Section 3.4.3] which is a special case of the general method in [1]. This algorithm might be of independent interest for other applications. Slight modifications of this basic algorithm are necessary in order to use it in the I-Max-Filter algorithm. They will be described later. In the following, we assume that n is a power of two. Adaption to the general case is simple by either rounding up to the next power of two and filling the array with $-\infty$ or by introducing a few case distinctions while initializing the data structure.

Consider a complete binary tree built on top of a so that the entries of a are the leaves (see level 0 in Figure 2). The idea is to store an array of prefix or suffix maxima with every internal node of the tree. Left successors store suffix maxima. Right successors store prefix maxima. The size of an array is proportional to the size of the subtree rooted at the corresponding node. To compute

the interval maximum $\max a[i..j]$, let v denote the least common ancestor of $a[i]$ and $a[j]$. Let u denote the left successor of v and let w denote the right successor of v . Let $u[i]$ denote the suffix maximum corresponding to leaf i in the suffix maxima array stored in u . Correspondingly, let $w[j]$ denote the prefix maximum corresponding to leaf j in the prefix maxima array stored in w . Then $\max a[i..j] = \max(u[i], w[j])$.

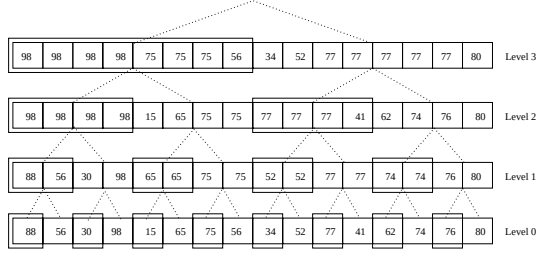


Fig. 2. Example of a layers array for interval maxima. The suffix sections are marked by an extra surrounding box.

We observed that this approach can be implemented in a very simple way using a $\log(n) \times n$ array preSuf . As can be seen in Figure 2, all suffix and prefix arrays in one layer can be assembled in one array as follows

$$\text{preSuf}[\ell][i] = \begin{cases} \max(a[2^\ell b..i]) & \text{if } b \text{ is odd} \\ \max(a[i..(2^\ell + 1)b - 1]) & \text{otherwise} \end{cases}$$

where $b = \lfloor i/2^\ell \rfloor$.

Furthermore, the interval boundaries can be used to index the arrays. We simply have $\max a[i..j] = \max(\text{preSuf}[\ell][i], \text{preSuf}[\ell][j])$ where $\ell = \text{msbPos}(i \oplus j)$; $\oplus$ is the bit-wise exclusive-or operation and $\text{msbPos}(x) = \lfloor \log_2 x \rfloor$ is equal to the position of the most significant nonzero bit of x (starting at 0). Some architectures have this operation in hardware¹; if not, $\text{msbPos}(x)$ can be stored in a table (of size n) and found by table lookup. Layer 0 is identical to a . A further optimization stores a pointer to the array $\text{preSuf}[\ell]$ in the layer table. As the computation is symmetric, we can conduct a table lookup with indices i, j without knowing whether $i < j$ or $j < i$.

To use this data structure for the I-Max-Filter algorithm we need a small modification since we are interested in maxima of the form $\max a[\min(i, j) + 1.. \max(i, j)]$ without knowing which of two endpoints is the smaller. Here we simply note that the approach still works if we redefine the suffix maxima to exclude the first entry, i.e., $\text{preSuf}[\ell][i] = \max(a[i + 1..(2^\ell + 1) \lfloor i/2^\ell \rfloor - 1])$ if $\lfloor i/2^\ell \rfloor$ is even.

¹ One trick is to use the exponent in a floating point representation of x .

```

(* Compute MST of  $G = (\{0, \dots, n-1\}, E)$  *)
Function I-Max-Filter-MST( $E$ ) : set of Edge
   $E' :=$  random sample from  $E$  of size  $\sqrt{mn}$ 
   $E'' :=$  JP-MST( $E'$ )
  Let  $\text{jpNum}[0..n-1]$  denote the order in which JP-MST added the nodes
  Initialize the table  $\text{preSuf}[0..\log n][0..n-1]$  as described in Section 2.2
  (* Filtering loop *)
  forall edges  $e = (u, v) \in E$  do
     $\ell := \text{msbPos}(\text{jpNum}[u] \oplus \text{jpNum}[v])$ 
    if  $w_e < \text{preSuf}[\ell][\text{jpNum}[u]]$  and  $w_e < \text{preSuf}[\ell][\text{jpNum}[v]]$  then add  $e$  to  $E''$ 
return JP-MST( $E''$ )

```

Fig. 3. The I-Max-Filter algorithm.

2.3 Putting the Pieces Together

Fig. 3 summarizes the I-Max-Filter algorithm and the following Theorem establishes its complexity.

Theorem 1. *The I-Max-Filter algorithm computes MSTs in expected time $mT_{\text{filter}} + \mathcal{O}(n \log n + \sqrt{nm})$ where T_{filter} is the time required to query the filter about one edge.*

In particular, if $m = \omega(n \log^2 n)$, the execution time is $(1 + o(1))mT_{\text{filter}}$.

Proof. Taking a sample can be implemented to run in constant time per sampled element. Running JP on the sample takes time $\mathcal{O}(n \log n + \sqrt{nm})$ if a Fibonacci heap (or another data structure with similar time bounds) is used for the priority queue. The lookup tables can be computed in time $\mathcal{O}(n \log n)$. The filtering loop takes time mT_{filter} .² By the sampling lemma explained in the introduction [8, Lemma 1], the expected number of edges in E'' is $n/\sqrt{n/m} = \sqrt{nm}$. Hence, running JP on E'' takes expected time $\mathcal{O}(n \log n + \sqrt{nm})$. Summing all the component execution times yields the claimed time bound.

From a theoretical point of view it is instructive to compare the number of edge weight comparisons needed to find an MST with the obvious lower bound of m . Also in this respect we are quite good for dense graphs because the filter algorithm performs at most two comparisons with each edge that is filtered out. In addition, an edge is already filtered out if the first comparison in Fig. 3 fails. Hence, a more detailed analysis might well show that we approach the lower bound of m for dense graphs.

² Note that it would be counterproductive to exempt the nodes in E' from filtering because this would require an extra test for each edge or we would have to compute $E - E'$ explicitly during sampling.

3 Experimental Evaluation

The objective of this section is to demonstrate that the I-Max-Filter algorithm is a serious contestant for the fastest MST algorithm for dense graphs ($m \gg n \log n$). We compare our implementation with a fast implementation of the JP algorithm. In [12] the execution time of the JP algorithm using different priority queues is compared and pairing heaps are found to be the fastest on dense graphs. We took the pairing heap from their code and combined it with a faster, array based graph representation.³ This implementation of JP consistently outperforms [12] and LEDA [11].

3.1 Graph Representations

One issue in comparing MST-algorithms for dense graphs is the underlying graph representation. The JP algorithm requires a representation that allows fast iteration over all edges that are adjacent to a given node. In a linked list implementation each edge resides in two linked lists; one for each incident node. In our *adjacency array* representation each edge is represented twice in an array with $2m$ entries such that the edges adjacent to each source node are stored contiguously. For each edge, the target node and weight is stored. In terms of space requirements, each source and each target is stored once, and only the weight is duplicated. A second array of size n holds for each node a pointer to the beginning of its adjacency array.

The I-Max-Filter algorithm, on the other hand, can be implemented to work well with any representation that allows sampling edges in time linear in the sample size and that allows fast iteration over all edges. In particular, it is sufficient to store each edge once. Our implementation for I-Max-Filter uses an array in which each edge appears once as (u, v) with $u < v$ and the edges are sorted by source node (u) .⁴ Only for the two small graphs for which the JP-algorithm is called it generates an adjacency array representation (see Fig. 3).

To get a fair comparison we decided that each algorithm gets the original input in its “favorite” representation. This decision favors JP because the conversion from an edge array to an adjacency array is much more expensive than vice versa. Furthermore, I-Max-Filter could run on the adjacency array representation with only a small overhead: during the sampling and filtering stages it would use the adjacency array while ignoring edges (u, v) with $u > v$.

³ The original implementation [12] uses linked lists which were quite appropriate at the time, when cache effects were less important.

⁴ These requirements could be dropped at very small cost. In particular, I-Max-Filter can work efficiently with a completely unsorted edge array or with an adjacency array representation that stores each edge only in one direction. The latter only needs space for $m + n$ node indices and m edge weights.

3.2 Filtering Access Pattern

Our implementation filters all edges stored with a node together so that it is likely that accesses to data associated with this node resides in cache.

Furthermore, the nodes are processed in the order given by JP order. This has the effect that only $\mathcal{O}(n)$ entries of the $\mathcal{O}(n \log n)$ lookup table entries need to be in cache at any time. In the results reported here (for graphs with up to 10,000 nodes), this access sequence resulted in a speedup of about 5 percent. For even larger graphs we have observed speedups of up to 11 % due to this optimization.

3.3 Implementation on Vector-Machines

A vector-machine has the capability to perform operations on vectors (instead of scalars) of some fixed size (in current vector-machines 256 or 512 elements) in one instruction. Vector-instructions typically include arithmetic and boolean operations, memory access instructions (consecutive, strided, and indirect), and special instructions like prefix-summation and minimum search. Vectorized memory accesses circumvent the cache. The filtering loop of Fig. 3 can readily be implemented on a vector-machine. The edges are stored consecutively in an array and can immediately be accessed in a vectorized loop; vectorized lookup of source and target vertices is possible by indirect memory access operations. For the filtering itself, bitwise exclusive or and two additional table lookups in the preSuf array are necessary. Using the prefix-summation capabilities, the edges that are not filtered out are stored consecutively in a new edge array. Also the construction of the preSuf data-structure can be vectorized. The only possibility for vectorization in the JP algorithm is the loop that scans and updates adjacent vertices of the vertex just added to the MST. We divide this loop into a scanning loop which collects the adjacent vertices for which a priority queue update is needed, and an update loop performing the actual priority queue updates. Using prefix-summation the scanning loop can immediately be vectorized. For the update there is little hope, unless a favorable data structure allowing simultaneous decrease-key operations can be devised.

3.4 Graph Types

Both algorithms, JP and I-Max-Filter were implemented in C++ and compiled using GNU g++ version 3.0.4 with optimization level -O6. We use a SUN-Fire-15000 server with 900 MHz UltraSPARC-III+ processors. Measurements on a Dell Precision 530 workstation with 1.7 GHz Intel P4 Xeon processors show similar results. The vector machine used is a NEC SX-6. The SX-6 has a memory bandwidth of 32GBytes/second, and (vector) peak-performance of 8GFlops.

We performed measurements with four different families of graphs, each with adjustable *edge density* $\rho = 2m/n(n-1)$. This includes all the families in [12] that admit dense inputs. A test instance is defined by three parameters: the

graph type, the number of nodes and the density of edges (the number of edges is computed from these parameters). Each reported result is the average of ten executions of the relevant algorithm; each on a different randomly generated graph with the given parameters. Furthermore, the I-Max-Filter algorithm is randomized because the sample graph is selected at random. Despite the randomization, the variance of the execution times within one test was consistently very small (less than 1 percent), hence we only plot the averages.

Worst-Case: $\rho \cdot n(n-1)/2$ edges are selected at random and the edges are assigned weights that cause JP to perform as many Decrease Key operations as possible [12].

Linear-Random: $\rho \cdot n(n-1)/2$ edges are selected at random. Each edge (u, v) is assigned the weight $w(u, v) = |u - v|$ where u and v are the integer IDs of the nodes.

Uniform-Random: $\rho \cdot n(n-1)/2$ edges are selected at random and each is assigned an edge weight which is selected uniformly at random.

Random-Geometric:[12] Nodes are random 2D points in a $1 \times y$ rectangle for some stretch factor $y > 0$. Edges are between nodes with Euclidean distance at most α and the weight of an edge is equal to the distance between its endpoints. The parameter α indirectly controls density whereas the stretch factor y allows us to interpolate between behavior similar to class Uniform-Random and behavior similar to class Linear-Random.

3.5 Results on Microprocessors

Fig. 4 shows execution times per edge on the SUN for the three graph families Worst-Case, Linear-Random and Uniform-Random for $n = 10000$ nodes and varying density. We can see that I-Max-Filter is up to 2.46 times faster than JP. This is not only for the “engineered” Worst-Case instances but also for Linear-Random graphs. The speedup is smaller for Uniform-Random graphs. On the Pentium 4 JP is even faster than I-Max-Filter on the Uniform-Random graphs. The reason is that for “average” inputs JP needs to perform only a sublinear number of decrease-key operations so that the part of code dominating the execution time of JP is scanning adjacency lists and comparing the weight of each edge with the distance of the target node from the current MST. There

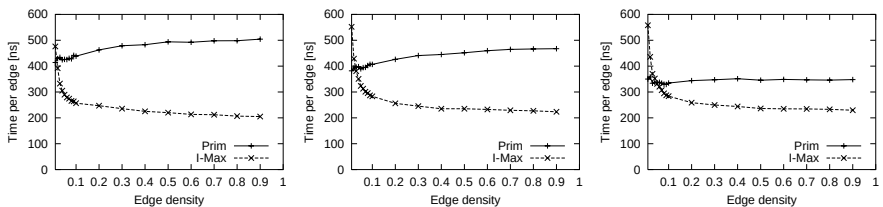


Fig. 4. Worst-Case, Linear-Random, and Uniform-Random graphs, 10000 nodes, SUN.

is no hope to be significantly faster than that. On the other hand, we observed a speedup of up to a factor of 3.35 on dense Worst-Case graphs. Hence, when we say that I-Max-Filter outperforms JP this is with respect to space consumption, simplicity of input conventions and worst-case performance guarantees rather than average case execution time.

On very sparse graphs, I-Max-Filter is up to two times slower than JP, because $\sqrt{mn} = \Theta(m)$ and as a result both the sample graph and the graph that remains after the filtering stage are not much smaller than the original graph. The runtime is therefore comparable to two runs of JP on the input.

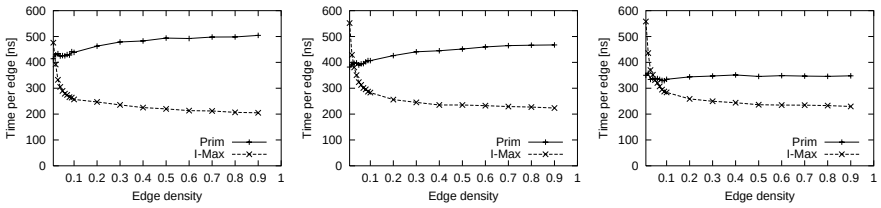


Fig. 5. Worst-Case, Linear-Random, and Uniform-Random graphs, 10000 nodes, NEC SX-6.

3.6 Results on a Vector Machine

Fig. 5 shows measurements on a NEC SX-6 vector computer analogous to the microprocessor results reported in Fig. 4.

For each of the two algorithms (JP and I-Max-Filter), runtimes per edge are plotted for scalar as well as vectorized version. The results of the scalar code show, once again, that JP is very fast on Uniform-Random graphs while I-Max-Filter is faster on the difficult graphs. In addition, we can see that on the “difficult” inputs I-Max-Filter benefits from vectorization more than JP which achieves a speedup of only factor 1.3. This is to be expected; JP becomes less vectorizable when many decrease key operations are performed, while the execution time of I-Max-Filter is dominated by the filtering stage, which in turn is not sensitive to the graph type. As a consequence, we see a speedup of up to 10 on the “difficult” graphs when comparing the vectorized versions of JP and I-Max-Filter.

3.7 Can JP Be Made Faster?

It is conceivable that the implementation of JP could be further improved using an even faster priority queue. Our implementation of JP uses the Pairing Heap variant that proved to be fastest in the comparative study of Moret and Shapiro [12]. How much can JP gain from an even faster heap? To investigate this we ran it with a best possible (theoretically impossible!) *perfect heap*, that

is, a heap in which both Decrease-Key and Delete-Minimum operations takes unit time. The *perfect heap* is implemented as an array, such that Decrease-Key takes constant time, and to simulate constant-time Delete-Minimum we simply stop the clock during this operation. Results for the worst-case graphs are shown in Fig. 6, which give both the run time break-down for I-Max-Filter, and the run time for I-Max-Filter with Pairing Heap and Perfect heap. The results show that I-Max-Filter is not very sensitive to the type of heap; its running time is dominated by the filtering stage which doesn't use the heap. JP is sensitive to the type of heap when running on graphs that incur many Decrease-Key operations, but not when it runs on a Uniform-Random graph (not shown here). All of this was to be expected, but in addition we see that I-Max-Filter is faster even when JP can access the heap almost for free and the only thing that takes time is traversing the nodes' adjacency lists.

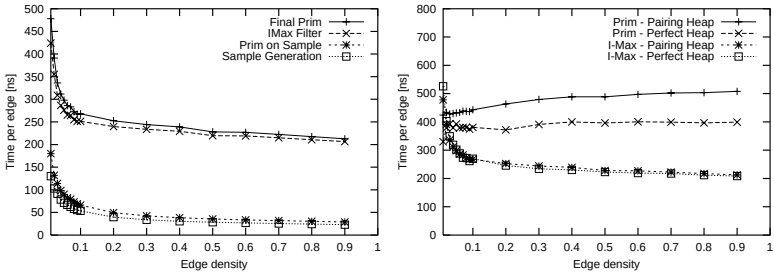


Fig. 6. Time break-down of I-Max-Filter (left). Pairing Heap vs. Perfect Heap (right). Worst-Case graph, 10,000 nodes, SUN.

4 Conclusions

We have seen that the cycle property can be practically useful to design improved MST algorithms for rather dense graphs. An open question is whether we can find improved practical algorithms for sparse graphs that use further ideas from the asymptotically best theoretical algorithms. Besides a component for filtering edges, these algorithms have a component for reducing the number of nodes based on Boruvka's [2,13] algorithm. Although this algorithm is conceptually simple, it seems unlikely that it is useful for internal memory algorithms on current machines. However node reduction has great potential for parallel and external-memory implementations.

References

1. N. Alon and B. Schieber. Optimal preprocessing for answering on-line product queries. Technical Report TR 71/87, Tel Aviv University, 1987.
2. O. Boruvka. O jistém problému minimálním. *Práce, Moravské Přírodovědecké Společnosti*, pages 1–58, 1926.
3. M. L. Fredman. On the efficiency of pairing heaps and related data structures. *Journal of the ACM*, 46(4):473–501, July 1999.
4. J. JáJá. *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.
5. V. Jarník. O jistém problému minimálním. *Práce Moravské Přírodovědecké Společnosti*, 6:57–63, 1930.
6. D. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. *Journal of the ACM*, 42(2):321–329, 1995.
7. V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18:263–270, 1997.
8. P. N. Klein and R. E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 9–15, 1994.
9. J. Komlós. Linear verification for spanning trees. In *25th annual Symposium on Foundations of Computer Science*, pages 201–206, 1984.
10. J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
11. K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
12. B. M. E. Moret and H. D. Shapiro. An empirical analysis of algorithms for constructing a minimum spanning tree. In *Workshop Algorithms and Data Structures (WADS)*, volume 519 of *Lecture LNCS*, pages 400–411. Springer, 1991.
13. J. Nešetřil, E. Milková, and H. Nešetřilová. Otakar Boruvka on minimum spanning tree problem: Translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1-3), 3–36, 2001.
14. S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *Journal of the ACM*, 49(1): 16–34, 2002.
15. R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, pages 1389–1401, 1957.

The Fractional Prize-Collecting Steiner Tree Problem on Trees^{*}

Extended Abstract

Gunnar W. Klau¹, Ivana Ljubić¹, Petra Mutzel¹, Ulrich Pferschy², and René Weiskircher¹

¹ Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Austria
`gunnar|ljubic|mutzel|weiskircher@ads.tuwien.ac.at`
² Department of Statistics and Operations Research
University of Graz, Austria, `pferschy@uni-graz.at`

Abstract. We consider the fractional prize-collecting Steiner tree problem on trees. This problem asks for a subtree T containing the root of a given tree $G = (V, E)$ maximizing the ratio of the vertex profits $\sum_{v \in V(T)} p(v)$ and the edge costs $\sum_{e \in E(T)} c(e)$ plus a fixed cost c_0 and arises in energy supply management. We experimentally compare three algorithms based on parametric search: the binary search method, Newton's method, and a new algorithm based on Megiddo's parametric search method. We show improved bounds on the running time for the latter two algorithms. The best theoretical worst case running time, namely $O(|V| \log |V|)$, is achieved by our new algorithm. A surprising result of our experiments is the fact that the simple Newton method is the clear winner of the tested algorithms.

1 Introduction

We consider a variant of the well-studied *Steiner tree problem in graphs*, namely the *prize-collecting Steiner tree problem*. This problem, where we want to find a subtree of a graph that maximizes an objective function that depends on the profits of the vertices and the costs of the edges, arises in the design of supply networks like district heating systems. It was first mentioned by Segev [14] where it appears as a special case of the node-weighted Steiner tree problem and is called the *Single Point Weighted Steiner Tree problem*. The author proves NP-hardness of the problem, presents integer linear programming formulations, and uses Lagrangean relaxation and heuristics to compute lower and upper bounds for these formulations, respectively. In [4], Duin and Volgenant relate the node-weighted (and thus also the prize-collecting) variant to the classical Steiner tree problem. They adapt reduction techniques and show how the

^{*} Partly supported by the Doctoral Scholarship Programme of the Austrian Academy of Sciences (DOC).

rooted prize-collecting Steiner tree problem can be transformed into the directed version of the classical Steiner tree problem.

In [6], Fischetti studies the facial structure of a generalization of the problem, the so called *Steiner arborescence problem*. Goemans studies the polyhedral structure of the node-weighted Steiner tree problem [7] and shows that his characterization is complete in case the input graph is series-parallel.

Approximation results are given by Bienstock *et al.* [1] and by Goemans and Williamson [8]; the latter present a purely combinatorial $O(n^2 \log n)$ -time primal-dual $(2 - \frac{1}{n-1})$ -approximation algorithm, where n denotes the number of vertices in the graph and the objective is to minimize the edge costs plus the prizes of the nodes not spanned. For the more realistic objective to maximize the sum of the profits minus the costs, Feigenbaum *et al.* [5] prove that it is NP-hard to approximate the problem to a constant factor.

In this paper, we look at the special case where the potential network is a tree and instead of the linear objective function, we look at the fractional version of the problem which maximizes the ratio of the sum of the profits and the sum of the (fixed and variable) costs.

Section 2 contains some preliminaries including the description of a linear time algorithm for optimizing the linear objective function. In Section 3, we present three different algorithms that use the parametric formulation: a binary search algorithm, Newton's method and our new variant based on Megiddo's method for parametric search. We show a worst case running time of Newton's method of $O(|V|^2)$, and of our new algorithm of $O(|V| \log |V|)$. In Section 4, we report on extensive computational experiments. Surprisingly for us, our experiments show that Newton's method, although having worst case running time of $O(|V|^2)$, outperforms the two other methods on our benchmark set. Finally, in Section 5 we summarize the results.

2 Preliminaries

In this section, we provide some basic definitions and describe a linear time algorithm for solving the linear version of the prize-collecting Steiner tree problem (PCST problem). A closely related dynamic programming algorithm can also be found in [15] (where trees with only node-weights are considered).

Let $G = (V, E)$ be an undirected graph, $r \in V$ a root vertex of G , $p : V \rightarrow \mathbb{R}^+ \cup \{0\}$ a profit function on the vertices, and $c : E \rightarrow \mathbb{R}^+ \cup \{0\}$ a cost function on the edges.

The *Fractional Prize Collecting Steiner Tree problem* (FPCST) consists of finding a connected subgraph $T = (V', E')$ of G with $r \in V'$ that maximizes the ratio of the profits and the costs:

$$\text{profit}(T) = \frac{\sum_{v \in V'} p(v)}{c_0 + \sum_{e \in E'} c(e)} ,$$

In the construction of a district heating network, the edges correspond to the potential pipes and the vertices to customers or forks in the pipe network. The

costs of the edges are the costs of the pipes, the profits of the vertices the revenue generated by the customers and the fixed cost c_0 the cost for building the heating plant.

The *Linear Prize Collecting Steiner Tree problem* (LPCST) consists of finding a connected subgraph $T = (V', E')$ of G with $r \in V'$ that maximizes the difference of the profits and the costs:

$$\text{profit}(T) = \sum_{v \in V'} p(v) - \sum_{e \in E'} c(e) .$$

Note that fixed costs are irrelevant if we optimize a linear objective function.

If $T = (V, E)$ is a tree with root r , then the function $\text{parent}(v)$ assigns every vertex $v \in V \setminus \{r\}$ a unique vertex u which is the vertex following v on the path from v to r . The *subtree* rooted at v consists of all vertices and edges reachable from v without passing the vertex $\text{parent}(v)$. The set $C(v)$ of *children* of v is the set that contains all vertices u with $\text{parent}(u) = v$. A subtree of T is *optimal*, if there is no other subtree of T with a higher profit. We recursively define a value $l(v)$ and a subtree $T(v)$ for each vertex $v \in V$ as

$$l(v) = p(v) + \sum_{u \in C(v)} \max\{0, l(u) - c(u, v)\} . \quad (1)$$

The subtree $T(v) = (V(v), E(v))$ with profit $l(v)$ is defined in the following way:

$$\begin{aligned} V(v) &= \{v\} \cup \bigcup_{u \in C(v)} \{V(u) \mid l(u) - c(u, v) \geq 0\} \\ E(v) &= \bigcup_{u \in C(v)} \{(u, v) \cup E(u) \mid l(u) - c(u, v) \geq 0\} . \end{aligned}$$

If $c(u, v) > l(u)$ for a vertex u with $\text{parent}(u) = v$ it does not pay off to include the subtree rooted at u via edge (u, v) (the only possible connection towards r), and we decide to cut off the edge (u, v) together with the corresponding subtree. This decision can be made locally, as soon as the value $l(u)$ is known. It is not hard to construct an algorithm for LPCST that uses these facts and runs in linear time (see [10] for details). The optimal subtree rooted at v is $T(v)$ with $l(v)$ as its profit (the correctness of this algorithm follows easily by induction).

When solving FPCST on trees, in contrast to the linear case, we cannot make local decisions anymore without looking at the whole problem. The following section presents the parametric formulation of the problem that allows us to decide in linear time if a given value t is smaller, equal, or greater than the value of an optimal solution of FPCST.

3 Algorithms Based on Parametric Formulation

To solve FPCST, we first formulate LPCST with an additional parameter. Then we show how this enables us to solve FPCST using our algorithm for LPCST.

The connection between a parametric formulation and the fractional version of the same problem has already been established by Dinkelbach [3].

Let $\mathcal{T}$ be the set of all connected subgraphs $T = (V', E')$ of G that contain the root. We are looking for a graph in $\mathcal{T}$ that maximizes the expression

$$\frac{\sum_{v \in V'} p(v)}{c_0 + \sum_{e \in E'} c(e)}.$$

Now consider the following function $o(t)$:

$$o: \mathbb{R}^+ \rightarrow \mathbb{R}, \quad o(t) = \max_{T=(V', E') \in \mathcal{T}} \sum_{v \in V'} p(v) - t(c_0 + \sum_{e \in E'} c(e)).$$

Let t^* be the value of the optimal solution of FPCST on G and $t \in \mathbb{R}$. Then we have:

$$o(t) = 0 \Leftrightarrow t = t^*, \quad o(t) < 0 \Leftrightarrow t > t^*, \quad o(t) > 0 \Leftrightarrow t < t^*.$$

Using the algorithm for LPCST, we can test for any t in linear time if it is smaller, equal, or greater than the optimal solution for FPCST. This fact can be used to construct different search algorithms that solve the problem.

There is also a geometric interpretation of our problem. Let $\mathcal{T}$ be again the set of all non-empty subtrees of G . Each $T = (V_T, E_T) \in \mathcal{T}$ defines a linear function $f_T: \mathbb{R}^+ \rightarrow \mathbb{R}$ in the following way:

$$f_T(t) = \sum_{v \in V_T} p(v) - t(c_0 + \sum_{e \in E_T} c(e)).$$

Since all vertex profits and edge costs are non-negative, and c_0 is positive, all these linear functions have negative slope. In this geometric interpretation, the function o defined above is the maximum of these functions. Hence it is a piecewise linear, convex, monotonously decreasing function. What we are looking for is the point where o crosses the x -axis. The functions f_T that contain this point correspond to optimal subtrees for the given profits and costs.

3.1 Binary Search

An easy way of building an algorithm for the FPCST problem that uses the parametric formulation of the previous section is binary search. We start with an interval (t_l, t_h) that contains t^* . Then we test the mid point t of this interval using the algorithm for the linear problem. This will give us either a proof that t equals t^* or a new upper or lower bound and will halve the size of the interval.

It is important to choose the right terminating conditions to achieve good performance. In our case, these conditions rely on the fact that $o(t)$ is the maximum of linear functions (see [10] for details). Since the running time of the algorithm depends to a great degree on the values for the profits and costs, a meaningful upper bound for the worst case running time that depends only on the size of the input graph cannot be given.

3.2 Newton's Method

We use the adaptation of Newton's iterative method described for example by Radzik [12]. Let $\mathcal{T}$ be the set of all subtrees of G that contain the root. We start with $t_0 = 0$. In iteration i , we compute

$$o(t_i) = \max_{T=(V', E') \in \mathcal{T}} \sum_{v \in V'} p(v) - t_i(c_0 + \sum_{e \in E'} c(e))$$

together with the optimal tree $T_i = (V_i, E_i)$ for parameter t_i using the linear algorithm from Section 2. As long as $o(t_i)$ is greater than 0, we compute t_{i+1} as the fractional objective value of T_i . So we have:

$$t_{i+1} = \frac{\sum_{v \in V_i} p(v)}{c_0 + \sum_{e \in E_i} c(e)}.$$

In the course of this algorithm, t_i increases monotonically until t^* is reached. Let l be the index with $t_l = t^*$. Radzik shows in [13] for general fractional optimization problems where all weights are non-negative that $l = O(p^2 \log^2 p)$ where p is the size of the problem (in our case the number of vertices of the problem graph G).

For our specific problem, we can prove a stronger bound for l :

Theorem 1. *Newton's method applied to the fractional prize-collecting Steiner tree problem with fixed costs takes at most $n+2$ iterations where n is the number of vertices of the input tree T .*

To proof the theorem, we show that for each iteration of Newton's method on our problem, there is an edge that was contained in the previous solution but is not contained in the current solution. This implies that the number of iterations is linear (see [10] for a detailed proof).

Since we can solve the problem for the linear objective function in linear time using the algorithm from Section 2, Newton's Method has a worst case running time of $O(|V|^2)$ for our problem.

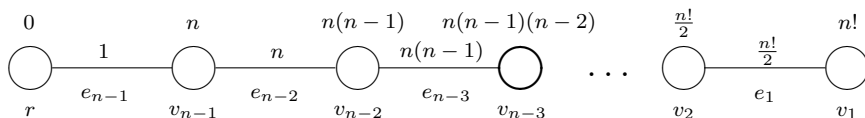


Fig. 1. Worst case example for Newton's Method. The edge costs and vertex profits are above the path while the names of the vertices and edges are below

Figure 1 shows an example where this worst case running time is reached. If we define the fixed costs $c_0 = 1$, we can show by a coarse estimation of the objective function value for each path starting at r that the solution of Newton's method shrinks only by one vertex in every iteration and that the

optimal solution is the root together with vertex v_{n-1} . Therefore, the algorithm executes $n - 1$ iterations and since each iteration has linear running time, the total running time of Newton's method on this example is $\Theta(n^2)$.

3.3 A New Algorithm Based on Megiddo's Parametric Search

In this section, we present our new algorithm for the FPCST problem which is a variant of parametric search introduced by Megiddo [11]. Furthermore, we suggest an improvement that guarantees a worst case running time of $O(n \log n)$ for any tree G with n vertices.

The idea of the basic algorithm is to simulate the execution of the algorithm A for LPCST on the unknown edge cost parameter t^* (the objective value of an optimal solution). During the simulation, we keep an interval (t_l, t_h) that contains t^* and that is initialized to $(0, \infty)$. Whenever A has to decide if a certain edge (u, v) is included in the solution, this decision is based on the evaluation of the maximum in (1) and depends on the root r_d of a linear function in t given by $l(u) - t \cdot c(u, v)$.

The decision is clear if r_d is outside (t_l, t_h) . Otherwise, we multiply all edge costs of the tree with r_d and execute A on the resulting problem. The sign of the linear objective function value $o(r_d)$ determines the decision (which enables us to continue the simulation of A) and r_d either becomes the new upper or lower bound of (t_l, t_h) .

There are two possibilities for the algorithm to terminate. The first is that one of the roots we test is t^* . In this case, we can stop without completing the simulation of A . If we have to simulate A completely, we end up with an interval for t^* . In this case, we perform depth first search on the edges that we have not cut during the simulation to obtain an optimal subtree.

Just as in the algorithm for the linear problem, our algorithm assigns labels to the vertices, but these labels are now linear functions that depend on the parameter t . The algorithm uses a copy G' of the problem tree G . In each phase, all leaves of G' are deleted after the necessary information has been propagated to the parents of the leaves. When the algorithm starts, the label of every vertex is set to the constant function equal to its profit. In the course of the algorithm, these labels change and will correspond to linear functions over the parameter t .

When we look at a certain leaf v with label $f_v(t)$ during a phase we compute the linear function $\tilde{f}_v(t) = f_v(t) - t \cdot c(e_v)$ where e_v is the edge incident to v . Let r_v be the root of $\tilde{f}_v(t)$. For all current leaves, we collect the values r_v , sort them and perform binary search on the roots using the linear algorithm to decide if the value t^* is smaller, greater, or equal than a certain root. Note that we do not have to include the roots in the binary search that are outside the current interval for t^* . If there are roots that are inside the current interval, we either find t^* or we end up with a smaller interval.

After the binary search, we know for each leaf v if its root r_v is smaller or greater than t^* (if it is equal, we have already found the solution and the algorithm has stopped). We delete all leaves whose root is smaller than t^* from G' . For all other leaves v , we add the function $\tilde{f}_v(t)$ to the label of its parent

and delete v , too. Now the next phase of the algorithm starts with the vertices that have become leaves because of the deletion of the current leaves (see [10] for a pseudo code).

The correctness of the algorithm follows from the general principle of Megiddo's method [11]. The running time of the algorithm is dominated by the calls to the linear algorithm. The binary search is performed by solving $O(\log(|B|))$ instances of LPCST with profits and costs determined by the parameter t . The set B is the set of leafs of the current working graph G' . Since it may happen that the graph contains only one leaf in every iteration (G may be a path) the number of iterations can be n . The worst case example for Newton's method in Section 3.2 is also a worst case example for this algorithm. Thus the overall running time of the algorithm is $O(|V|^2)$.

Improvement Through Path Contraction. If there is no vertex in G with degree two, our algorithm already has a running time of $O(n \log n)$ for a tree with n vertices: In this case we delete at least half the vertices of the graph in every iteration by deleting all leaves. It will follow from the proof of Theorem 2 that this property is sufficient for the improved running time.

We will remove the remaining obstacles in the graph, namely vertices of degree two, by performing a reduction of all paths in the tree. This must be done in every iteration since the removal of all leaves at the end of the previous iteration may generate new paths. The idea of the reduction is based on the fact that the subtree situated at the end of a path can only contribute to the optimal solution if the complete path is also included. Otherwise, only a connected subset of the path can be in the optimal solution.

More formally, a subset of V is a path denoted by $P := \{v_0, v_1, \dots, v_m, v_{m+1}\}$ if v_0 has degree greater two or is the root, v_{m+1} does not have degree two and all other vertices are of degree two. To fix the orientation we assume that v_0 is included in the path from v_1 to r . Since we want to contract the m vertices of the path to a single vertex, trivial cases can be excluded by assuming $m \geq 2$. In an optimal solution either there exists a vertex $v_q \in P$ such that $v_1, \dots, v_q$ are the only vertices of P in the solution, or P is completely contained in the solution and connects a possible subtree rooted at v_{m+1} to r .

The procedure CONTRACTPATH (see Algorithm 1) determines the best possible candidate for v_q and contracts the path by adding an artificial edge from v_0 to v_q with cost equal to the value of the complete subpath including $v_1, \dots, v_{q-1}$, and a second artificial edge from v_q to v_{m+1} that models the cost of traversing the vertices $v_{q+1}, \dots, v_m$. The path contraction is invoked at the beginning of every iteration in our algorithm for FPCST.

The main theoretical result of this paper is stated in the following theorem:

Theorem 2. *The running time of Algorithm the algorithm with CONTRACTPATH is in $O(n \log n)$.*

Proof. (Sketch) To find v_q , we need to compute the maximum of m linear functions, which can be done in time $O(m \log m)$ (see [2] for a proof). The resulting

```

Data : A labeled tree  $T = (V, E)$  with fixed root  $r$ ;
        a path in  $T$   $v_0, v_1, \dots, v_m, v_{m+1}$ ,  $m > 2$ 
Result : A labeled tree  $T = (V, E)$  with fixed root  $r$ 
 $end[1] = 0$ ;
for  $j = 1$  to  $m$  do
     $end[j] := end[j - 1] + l(v_j) + c(v_{j-1}, v_j)$ ;
end
 $f(t) = \max_{j=1}^m end[j]$ ;
 $B = \{t \in (t_l, t_h) \mid t \text{ is breakpoint of } f(t)\} \cup \{t_l, t_h\}$ ;
Perform binary search on  $B$  using the modified linear algorithm and update  $t_l$ 
and  $t_h$ ;
choose  $q$  s.t.  $end[q] = f(t)$  for  $t \in (t_l, t_h)$ ;
 $c(v_0, v_q) := \sum_{k=1}^{q-1} (l(v_k) + c(v_{k-1}, v_k)) + c(v_{q-1}, v_q)$ ;
 $c(v_q, v_{m+1}) = \sum_{k=q+1}^m (l(v_k) + c(v_{k-1}, v_k)) + c(v_m, v_{m+1})$ ;
Remove vertices  $v_1, \dots, v_{q-1}, v_{q+1}, \dots, v_m$  from  $T$ ;

```

Algorithm 1: Algorithm CONTRACTPATH to remove all nontrivial paths from a tree

piecewise linear function has at most m breakpoints. In every iteration there is a number of breakpoints from CONTRACTPATH and a number of leaves with corresponding root values to be considered. We use binary search in each iteration to find a new interval (t_l, t_h) including neither breakpoints nor roots thus resolving the selection of v_q and the final decision on all leaves.

If k is the size of the graph at the beginning of an iteration, then the binary search performs a logarithmic number of calls to the algorithm that solves LPCST. Therefore, a single iteration takes time $O(k \log k)$. It can be shown that applying the procedure CONTRACTPATH to every non trivial path guarantees that our algorithm together with CONTRACTPATH deletes at least one third of the vertices in each iteration. Since the size of the graph is reduced by a constant fraction after each iteration, the total running time sums up to $O(n \log n)$. See [10] for a detailed proof. $\square$

4 Computational Experiments

We generated two different test sets of graphs to test the performance of the algorithms presented in Section 3. The first set consists of randomly generated trees where every vertex has at most two children while the second set contains random trees where each vertex can have up to ten children. In both sets, the cost of each edge and the profit of each vertex is a random integer from the set $\{1, 2, \dots, 10,000\}$. Both sets contain 100 trees for each number of vertices from 1,000 to 10,000 in steps of 500 vertices. The fixed costs for all problem instances has been chosen as 1,000 times the number of vertices in the graph. This produces solutions containing around 50% of all vertices for the graphs where each vertex has at most 10 children. For the graphs where each vertex

has at most two children, the percentage is around 35%. To execute the three algorithms on the test sets as a documented and repeatable experiment and for analyzing the results, we used the tool set **ExpLab** [9].

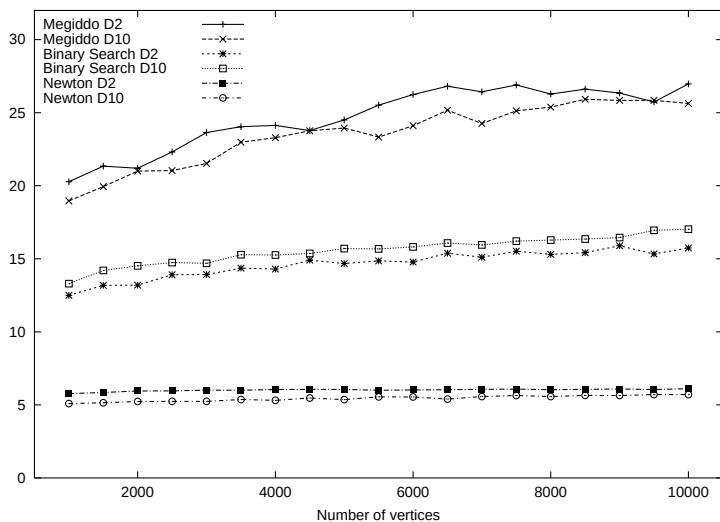


Fig. 2. The average number of calls to the linear algorithm executed by the three algorithms on the benchmark set with maximum degree 2 and maximum degree 10

Figure 2 shows the average number of calls over all trees with the same number of vertices for the three algorithms and the two benchmark sets. The number of calls grows very slowly with the size of the graphs for all three algorithms. In fact, the number of calls barely grows with the number of vertices in the graph for Newton’s method.

Our variant of Megiddo’s method needs more calls than the other two methods. For the leaves of the tree, the algorithm behaves just like binary search. The reason why the number of calls is higher than for binary search is that our new algorithm not only executes calls at the leaf level but also higher up in the tree. These are usually very few and not on every level. So on a level where additional calls have to be made, there are usually only one or two open decisions. Therefore, the binary search in our new algorithm can not effectively be used except at the leaf level. Because of this fact, the pure binary search algorithm can “jump” over some decisions that parametric search has to make on higher levels.

The reason why Newton’s method needs fewer calls than the binary search method is the random nature of our problem instances. Binary search starts with a provable upper bound for t^* which in our case is the sum of all vertex profits divided by the fixed costs. This upper bound is far away from the objective value of the optimal solution. After the first iteration of Newton’s method, the

value t is the objective function value of the whole tree. This value is a good lower bound for the optimal solution because the profits and costs are random and with the fixed costs we have chosen, the optimal tree contained 35-50% of all vertices. Therefore, Newton's method needs only a small number of steps to reach the optimal solution and the number of calls grows only very slowly with the size of the graphs.

Figure 3 shows that the number of calls to the linear algorithm determines the running time: our new algorithm is the slowest and Newton's method the fastest. The running times grow slightly faster than linear with the size of the graphs. Since each call to the algorithm for the linear problem needs linear time, the fact that the number of calls grows with the size of the graph (albeit very slowly) is the reason for this behavior. We executed the experiments on a PC with a 2.8 GHz Intel Processor with 2GB of memory running Linux. Even for the graphs with 10,000 vertices, the problems can be solved in less than 1.8 seconds.

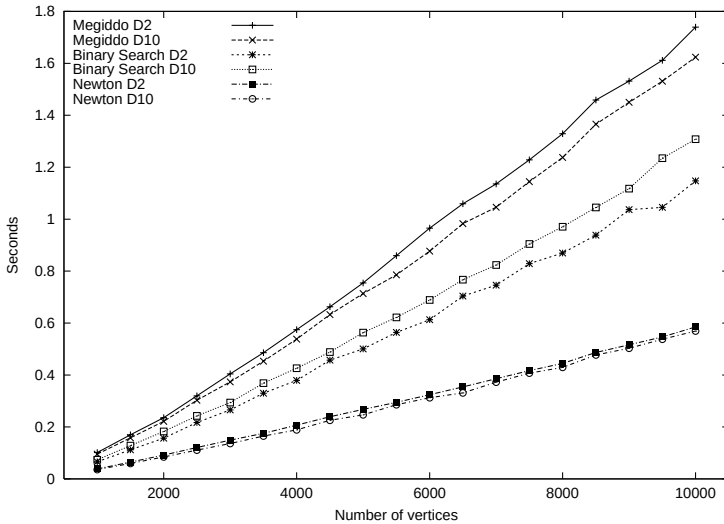


Fig. 3. The average time used by the three algorithms on the two benchmark sets

We also executed an experiment where we used only the 100 graphs of the test set with maximum degree 10 that have 10,000 vertices. We increased the fixed costs c_0 exponentially and ran all three algorithms on the 100 graphs for each value of c_0 . We started with $c_0 = 100$ (where the solution contained only a few vertices) and multiplied the fixed costs by 10 until we arrived at 10^{11} (where the optimal solution consisted almost always of the whole tree).

Figure 4 shows how the time needed by the three algorithms depends on fixed costs. It is remarkable that for small fixed costs, binary search is faster than Newton's method but for fixed costs of more than 10,000, Newton's method is

faster. The reason is the same we have already given for the better performance of Newton's method in our first experiments. For large fixed costs, the percentage of the vertices contained in an optimal solution rises and so the value of the first solution that Newton's method tests, which is the value of the whole graph, is already very close to the optimal value. Binary search has to approach the optimum solution from the provable upper bound for the objective function value which is far away from the optimal solution when this solution is large and therefore contains many edges.

Parametric search is not much slower than binary search for high fixed costs. As the plot shows, the reason is not that parametric search performs significantly better for higher fixed costs but that the performance of binary search deteriorates for the reasons given in the last paragraph.

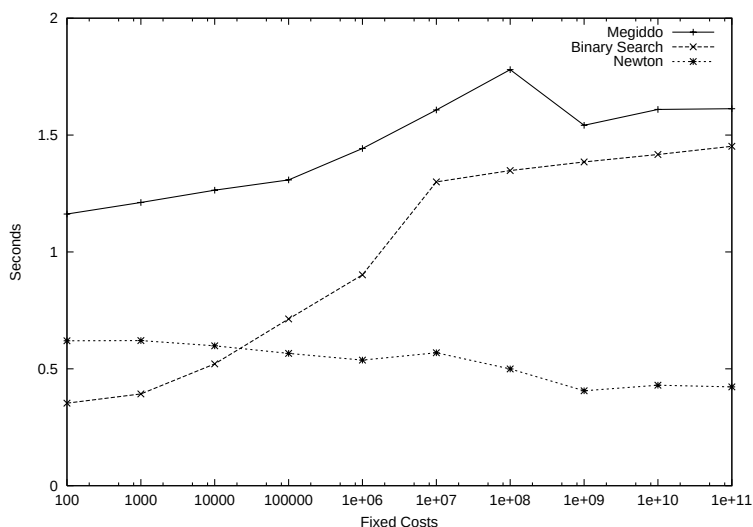


Fig. 4. Time used by the three algorithms for growing fixed costs (logarithmic x-axis)

5 Conclusions

In this paper, we have presented three algorithms for solving the fractional prize-collecting Steiner tree problem (PCST problem) on trees $G = (V, E)$. We have shown that Newton's algorithm has a worst case running time of $O(|V|^2)$. We have also presented a variant of parametric search and proved that the worst case running time of this new algorithm is $O(|V| \log |V|)$. Our computational results show that Newton's method performs best on randomly generated problems while a simple binary search approach and our new method are considerably slower. For all three algorithms, the running time grows slightly faster than linear with the size of our test instances.

Acknowledgments. We thank Günter Rote and Laurence Wolsey for giving us useful pointers to the literature.

References

1. D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.
2. J. D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998.
3. W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13:492–498, 1967.
4. C. W. Duin and A. Volgenant. Some generalizations of the Steiner problem in graphs. *Networks*, 17(2):353–364, 1987.
5. J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, 2001.
6. M. Fischetti. Facets of two Steiner arborescence polyhedra. *Mathematical Programming*, 51:401–419, 1991.
7. M. X. Goemans. The Steiner tree polytope and related polyhedra. *Mathematical Programming*, 63:157–182, 1994.
8. M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191. P. W. S. Publishing Co., 1996.
9. S. Hert, L. Kettner, T. Polzin, and G. Schäfer. Explab. <http://explab.sourceforge.net>, 2002.
10. G. Klau, I. Ljubić, P. Mutzel, U. Pferschy, and R. Weiskircher. The fractional prize-collecting Steiner tree problem on trees. Technical Report TR-186-1-03-01, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2003.
11. N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979.
12. T. Radzik. Newton’s method for fractional combinatorial optimization. In *Proceedings of 33rd Annual Symposium on Foundations of Computer Science*, pages 659–669, 1992.
13. T. Radzik. Fractional combinatorial optimization. In D. Z. Du and P. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 429–478. Kluwer, 1998.
14. A. Segev. The node-weighted Steiner tree problem. *Networks*, 17:1–17, 1987.
15. L. A. Wolsey. *Integer Programming*. John Wiley, New York, 1998.

Algorithms and Experiments for the Webgraph^{*}

Luigi Laura¹, Stefano Leonardi¹, Stefano Millozzi¹, Ulrich Meyer², and
Jop F. Sibeyn³

¹ Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via
Salaria 113, 00198 Roma Italy. {laura,leon,millozzi}@dis.uniroma1.it

² Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken,
Germany. umeyer@mpi-sb.mpg.de

³ Halle University, Institute of Computer Science, Von-Seckendorff-Platz 1, 06120
Halle Germany. jopsi@informatik.uni-halle.de

Abstract. In this paper we present an experimental study of the properties of web graphs. We study a large crawl from 2001 of 200M pages and about 1.4 billion edges made available by the WebBase project at Stanford [19], and synthetic graphs obtained by the large scale simulation of stochastic graph models for the Webgraph. This work has required the development and the use of external and semi-external algorithms for computing properties of massive graphs, and for the large scale simulation of stochastic graph models. We report our experimental findings on the topological properties of such graphs, describe the algorithmic tools developed within this project and report the experiments on their time performance.

1 Introduction

The *Webgraph* is the graph whose nodes are (static) web pages and edges are (directed) hyperlinks among them. The *Webgraph* has been the subject of a large interest in the scientific community. The reason of such large interest is primarily given to search engine technologies. Remarkable examples are the algorithms for ranking pages such as PageRank [4] and HITS [9].

A large amount of research has recently been focused on studying the properties of the Webgraph by collecting and measuring samples spanning a good share of the whole Web. A second important research line has been the development of stochastic models generating graphs that capture the properties of the Web. This research work also poses several algorithmic challenges. It requires to develop algorithmic tools to compute topological properties on graphs of several billion edges.

^{*} Partially supported by the Future and Emerging Technologies programme of the EU under contracts number IST-2001-33555 COSIN "Co-evolution and Self-organization in Dynamical Network" and IST-1999-14186 ALCOM-FT "Algorithms and Complexity in Future Technologies", and by the Italian research project ALINWEB: "Algoritmica per Internet e per il Web", MIUR – Programmi di Ricerca di Rilevante Interesse Nazionale.

The Webgraph has shown the ubiquitous presence of power law distributions, a typical signature of scale-free properties. Barabasi and Albert [3] and Kumar et al [11] suggested that the in-degree of the Webgraph follow a *power-law* distribution. Later experiments by Broder et al. [5] on a crawl of 200M pages from 1999 by Altavista confirmed it as a basic property: the probability that the in-degree of a vertex is i is distributed as $Pr_u[\text{in-degree}(u)=i] \propto 1/i^\gamma$, for $\gamma \approx 2.1$. In [5] the out-degree of a vertex was also shown to be distributed according to a power law with exponent roughly equal to 2.7 with exception of the initial segment of the distribution. The number of edges observed in the several samples of the Webgraph is about equal to 7 times the number of vertices.

Broder et. al. [5] also presented a fascinating picture of the Web's macroscopic structure: a *bow-tie* shape with a core made by a large strongly connected component (SCC) of about 28% of the vertices. A surprising number of specific topological structures such as bipartite cliques of relatively small size has been observed in [11]. The study of such structures is aimed to trace the emergence of hidden *cyber-communities*. A bipartite clique is interpreted as a core of such a community, defined by a set of fans, each fan pointing to a set of centers/authorities for a given subject, and a set of centers, each pointed by all the fans. Over 100,000 such communities have been recognized [11] on a sample of 200M pages on a crawl from Alexa of 1997.

The Google search engine is based on the popular PageRank algorithm first introduced by Brin and Page [4]. The PageRank distribution has a simple interpretation in terms of a random walk in the Webgraph. Assume the walk has reached page p . The walk then continues either by following with probability $1-c$ a random link in the current page, or by jumping with probability c to a random page. The correlation between the distribution of PageRank and in-degree has been recently studied in a work of Pandurangan, Raghavan and Upfal [15]. They show by analyzing a sample of 100,000 pages of the brown.edu domain that PageRank is distributed with a power law of exponent 2.1. This exactly matches the in-degree distribution, but very surprisingly it is observed very little correlation between these quantities, i.e., pages with high in-degree may have low PageRank.

The topological properties observed in the WebGraph, as for instance the in-degree distribution, cannot be found in the traditional random graph model of Erdős and Rényi (ER) [7]. Moreover, the ER model is a static model, while the Webgraph evolves over time when new pages are published or are removed from the Web.

Albert, Barabasi and Jeong [1] initiated the study of evolving networks by presenting a model in which at every discrete time step a new vertex is inserted in the graph. The new vertex connects to a constant number of previously inserted vertices chosen according to the *preferential attachment* rule, i.e. with probability proportional to the in-degree. This model shows a power law distribution over the in-degree of the vertices with exponent roughly 2 when the number of edges that connect every vertex to the graph is 7. In the following sections we refer to this model as the *Evolving Network (EN)* model.

The Copying model has been later proposed by Kumar et al. [10] to explain other relevant properties observed in the Webgraph. For every new vertex entering the graph a prototype vertex p is selected at random. A constant number d of links connect the new vertex to previously inserted vertices. The model is parameterized on a *copying factor* α . The end-point of a link is either copied with probability α from a link of the prototype vertex p , or it is selected at random with probability $1 - \alpha$. The copying event aims to model the formation of a large number of bipartite cliques in the Webgraph. In our experimental study we consider the *linear*[10] version of this model, and we refer to it simply as the *Copying* model.

More models of the Webgraph are presented by Pennock et al. [16], Caldarelli et al. [12], Panduragan, Raghavan and Upfal [15], Cooper and Frieze [6]. Mitzenmacher [14] presents an excellent survey of generative models for power-law distributions. Bollobás and Riordan [2] study vulnerability and robustness of scale-free random graphs. Most of the models presented in the literature generate graphs without cycles. Albert et al. [1] amongst others proposed to rewire part of the edges introduced in previous steps to induce links in the graphs.

Outline of the paper. We present an extensive study of the statistical properties of the Webgraph by analyzing a crawl of about 200M pages collected in 2001 by the WebBase project at Stanford [19] and made available for our study. The experimental findings on the structure of the WebBase crawl are presented in Section 2.

We also report new properties of some stochastic graph models for the Webgraph presented in the literature. In particular, in Section 3, we study the distribution of the size and of the number of strongly connected components.

This work has required the development of semi-external memory [18] algorithms for computing disjoint bipartite cliques of small size, external memory algorithms [18] based on the ideas of [8] for computing PageRank, and the large scale simulation of stochastic graph models. Moreover, we use the semi-external algorithm developed in [17] for computing Strongly Connected Components. The algorithms and the experimental evaluation of their time performances are presented in Section 4. A detailed description of the software tools developed within this project can be found [13].

2 Analysis of the WebBase Crawl

We conducted our experiments on a 200M nodes crawl collected from the WebBase project at Stanford [19] in 2001.

The in-degree distribution follows a power law with $\gamma = 2.1$. This confirms the observations done on the crawl of 1997 from Alexa [11], the crawl of 1999 from Altavista [5] and the notredame.edu domain [3].

In Figure 1 the out-degree distribution of the WebBase crawl is shown. While the in-degree distribution is fitted with a power law, the out-degree is not, even for the final segment of the distribution. A deviation from a power law for the initial segment of the distribution was already observed in the Altavista crawl [5].

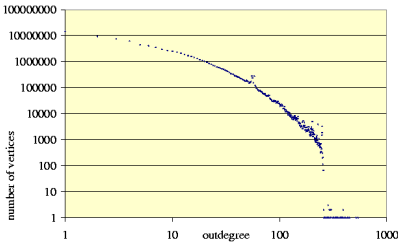


Fig. 1. Out-degree distribution of the Web Base crawl

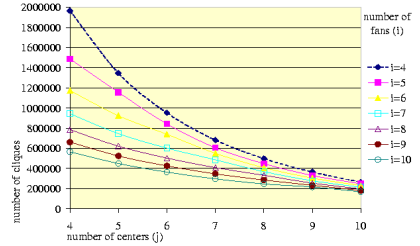


Fig. 2. The number of bipartite cliques (i, j) in the Web Base crawl

We computed the PageRank distribution of the WebBase crawl. Here, we confirm the observation of [15] by showing this quantity distributed according to a power-law with exponent $\gamma = 2.109$. We also computed the statistical correlation between PageRank and in-degree. We obtained a value of $-5.2E-6$, on a range of variation in $[-1, 1]$ from negative to positive correlation. This confirms on much larger scale the observation done by [15] on the brown.edu domain of 100,000 pages that the correlation between the two measures is not significant.

In Figure 2 the graphic of the distribution of the number of bipartite cliques (i, j) , with $i, j = 1, \dots, 10$ is shown. The shape of the graphic follows that one presented by Kumar et al. [11] for the 200M crawl by Alexa. However, we detect a number of bipartite cliques of size $(4, j)$ that differs from the crawl from Alexa for more than one order of magnitude. A possible (and quite natural) explanation is that the number of *cyber-communities* has consistently increased from 1997 to 2001. A second possible explanation is that our algorithm for finding disjoint bipartite cliques, which is explained in 4.1, is more efficient than the one implemented in [11].

3 Strongly Connected Components

Broder et al. [5] identified a very large strongly connected component of about 28% of the entire crawl. The Evolving Network and the Copying model do not contain cycles and hence not even a single strongly connected component. We therefore modified the EN and the Copying model by rewiring a share of the edges. The process consists of adding edges whose end-points are chosen at random. The experiment consisted in rewiring a number of edges ranging from 1% to 300% of the number of vertices in the graph. Recall that these graphs contain 7 times as many edges as the vertices.

The most remarkable observation is that, differently from the Erdős-Renyi model, we do not observe any threshold phenomenon in the emerging of a large SCC. This is due to the existence of a number of vertices of high in-degree in a graph with in-degree distributed according to a power law. Similar conclusions are also formally obtained for scale-free undirected graphs by Bollobas and Riordan [2]. In a classical random graph, it is observed the emerging of a giant

connected component when the number of edges grows over a threshold that is slightly more than linear in the number of vertices. We observe the size of the largest SCC to increase smoothly with the number of edges that are rewired up to span a big part of the graph. We also observe that the number of SCCs decreases smoothly with the increase of the percentage of rewired edges. This can be observed in Figure 3 for the Copying model on a graph of 10M vertices. A similar phenomenon is observed for the Evolving Network model.

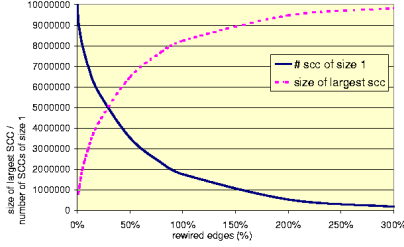


Fig. 3. Number and size of SCCs – (Copying Model)

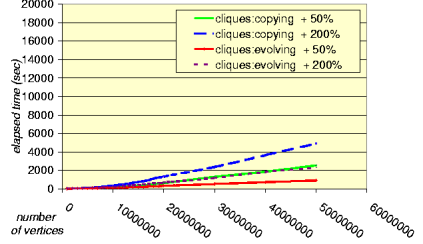


Fig. 4. The time performance of the computation of disjoint cliques (4, 4)

Devising strongly connected components in a graph stored on secondary memory is a non-trivial task for which we used a semi-external algorithm developed in [17]. This algorithm together with its time performance is described in Section 4.

4 Algorithms for Analyzing and Generating Web Graphs

In this section we present the external and semi-external memory algorithms we developed and used in this project for analyzing massive Webgraphs and their time performance. Moreover, we will present some of the algorithmic issues related to the large scale simulation of stochastic graph models.

For measuring the time performance of the algorithms we have generated graphs according to the Copying and the Evolving Network model. In particular, we have generated graphs of size ranging from 100,000 to 50M vertices with average degree 7, and rewired a number of edges equal to 50% and 200% of the vertices. The presence of cycles is fundamental for both computing SCCs and PageRank. This range of variation is sufficient to assess the asymptotic behavior of the time performance of the algorithms. In our time analysis we computed disjoint bipartite cliques of size (4, 4), the size for which the computational task is more difficult.

The analysis of the time complexity of the algorithms has been performed by restricting the main memory to 256MB for computing disjoint bipartite cliques and PageRank. For computing strongly connected components, we have used

1GB of main memory to store a graph of 50M vertices with 12.375 bytes per vertex. Figures 4, 5 and 6 show the respective plots. The efficiency of these external memory algorithms is shown by the linear growth of the time performance whenever the graph does not fit in main memory. More details about the data structures used in the implementation of the algorithms are given later in the section.

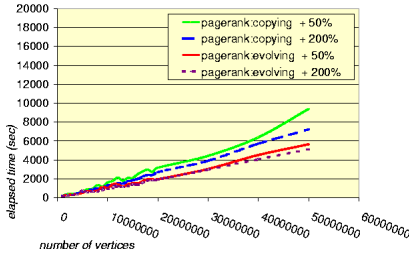


Fig. 5. The time performance of the computation of PageRank

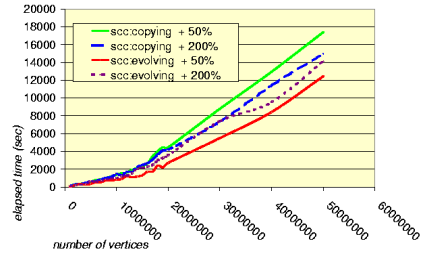


Fig. 6. The time performance of the computation of SCCs

4.1 Disjoint Bipartite Cliques

In [11] an algorithm for enumerating disjoint bipartite cliques (i, j) of size at most 10 has been presented, with i being the fan vertices on the left side and j being the center vertices on the right side. The algorithm proposed by Kumar et al. [11] is composed of a pruning phase that consistently reduces the size of the graph in order to store it in main memory. A second phase enumerates all bipartite cliques of the graph. A final phase selects a set of bipartite cliques that form the solution. Every time a new clique is selected, all intersecting cliques are discarded. Two cliques are intersecting if they have a common fan or a common center. A vertex can then appear as a fan in a first clique and as a center in a second clique.

In the following, we describe our semi-external heuristic algorithm for computing disjoint bipartite cliques. The algorithm searches bipartite cliques of a specific size (i, j) .

Two n -bit arrays *Fan* and *Center*, stored in main memory, indicate with $Fan(v) = 1$ and $Center(v) = 1$ whether fan v or center v has been removed from the graph. We denote by $I(v)$ and $O(v)$ the list of predecessors and successors of vertex v . Furthermore, let $\tilde{I}(v)$ be the set of predecessors of vertex v with $Fan(\cdot) = 0$, and let $\tilde{O}(v)$ the set of successors of vertex v with $Center(\cdot) = 0$. Finally, let $T[i]$ be the first i vertices of an ordered set T .

We first outline the idea underlying the algorithm. Consider a fan vertex v with at least j successors with $Center(\cdot) = 0$, and enumerate all size j subsets of $\tilde{O}(v)$. Let S be one such subset of j vertices. If $|\cap_{u \in S} I(u)| \geq i$ then we have

detected an (i, j) clique. We remove the fan and the center vertices of this clique from the graph. If the graph is not entirely stored in main memory, the algorithm has to access the disk for every retrieval of the list of predecessors of a vertex of $O(v)$. Once the exploration of a vertex has been completed, the algorithm moves to consider another fan vertex.

In our semi-external implementation, the graph is stored on secondary memory in a number of blocks. Every block b , $b = 1, \dots, \lceil N/B \rceil$, contains the list of successors and the list of predecessors of B vertices of the graph. Denote by $b(v)$ the block containing vertex v , and by $B(b)$ the vertices of block b . We start by analyzing the fan vertices from the first block and proceed until the last block. The block currently under examination is moved to main memory. Once the last block has been examined, the exploration continues from the first block.

We start the analysis of a vertex v when block $b(v)$ is moved to main memory for the first time. We start considering all subsets S of $\tilde{O}(v)$ formed by vertices of block $b(v)$. However, we also have to consider those subsets of $\tilde{O}(v)$ containing vertices of other blocks, for which the list of predecessors is not available in main memory. For this purpose, consider the next block b' that will be examined that contains a vertex of $\tilde{O}(v)$. We store $\tilde{O}(v)$ and the lists of predecessors of the vertices of $\tilde{O}(v) \cap B(b)$ into an auxiliary file $A(b')$ associated with vertex b' . We actually buffer the access to the auxiliary files. Once the buffer of block b reaches a given size, this is moved to the corresponding auxiliary file $A(b)$. In the following we abuse notation by denoting with $A(b)$ also the set of fan vertices v whose exploration will continue with block b .

When a block b is moved to main memory, we first seek to continue the exploration from the vertices of $A(b)$. If the exploration of a vertex v in $A(b)$ cannot be completed within block b , the list of predecessors of the vertices of $\tilde{O}(v)$ in blocks from $b(v)$ to block b are stored into the auxiliary file of the next block b' containing a vertex of $\tilde{O}(v)$. We then move to analyze the vertices $B(b)$ of the block. We keep on doing this till all fan and center vertices have been removed from the graph. It is rather simple to see that every block is moved to main memory at most twice.

The core algorithm is preceded by two pruning phases. The first phase removes vertices of high degree as suggested in [11] since the objective is to detect cores of hidden communities. In a second phase, we remove vertices that cannot be selected as fans or centers of an (i, j) clique.

Phase I. Remove all fans v with $|O(v)| \geq 50$ and all centers v with $|I(v)| \geq 50$.

Phase II. Remove all fans v with $|\tilde{O}(v)| < i$ and all centers with $|\tilde{I}(v)| < j$.

When a fan or a center is removed in Phase II, the in-degree or the out-degree of a vertex is also reduced and this can lead to further removal of vertices. Phase II is carried on few times till only few vertices are removed. Phases I and II can be easily executed in a streaming fashion as described in [11]. After the pruning phase, the graph of about 200M vertices is reduced to about 120M vertices. About 65M of the 80M vertices that are pruned belong to the border of the graph, i.e. they have in-degree 1 and out-degree 0.

We then describe the algorithm to detect disjoint bipartite cliques.

Phase III.

1. While there is a fan vertex v with $Fan(v) = 0$
2. Move to main memory the next block b to be examined.
3. For every vertex $v \in A(b) \cup B(b)$ such that $|\tilde{O}(v)| \geq j$
 - 3.1 For every subset S of size j of $\tilde{O}(v)$, with the list of predecessors of vertices in S stored either in the auxiliary file $A(b)$ or in block b :
 - 3.2 If $|T = \cap_{u \in S} \tilde{I}(u)| \geq i$ then
 - 3.2.1 output clique $(T[i], S)$
 - 3.2.2 set $Fan(\cdot) = 1$ for all vertices of $T[i]$
 - 3.2.3 set $Center(\cdot) = 1$ for all vertices of S

Figure 4 shows the time performance of the algorithm for detecting disjoint bipartite cliques of size $(4, 4)$ on a system with 256 MB. 70 MB are used by the operating system, including operating system's cache. We reserve 20MB for the buffers of the auxiliary files. We maintain 2 bit information $Fan(\cdot)$ and $Center(\cdot)$ for every vertex, and store two 8bytes pointer to the list of successors and the list of predecessors of every vertex. Every vertex in the list of adjacent vertices requires 4 bytes. The graph after the pruning has average out/in 8.75. Therefore, on the average, we need about $0.25N + B(2 \times 8 + 17.5 \times 4)$ bytes for a graph of N vertices and block size B . For a graph of 50M vertices this results in a block size of 1.68M vertices. We performed our experiments with a block size of 1M vertices. We can observe the time performance to converge to a linear function for graphs larger than this size.

4.2 PageRank

The computation of PageRank is expressed in matrix notation as follows. Let N be the number of vertices of the graph and let $n(j)$ be the out-degree of vertex j . Denote by M the square matrix whose entry M_{ij} has value $1/n(j)$ if there is a link from vertex j to vertex i . Denote by $[\frac{1}{N}]_{N \times N}$ the square matrix of size $N \times N$ with entries $\frac{1}{N}$. Vector *Rank* stores the value of PageRank computed for the N vertices. A matrix M' is then derived by adding transition edges of probability $(1 - c)/N$ between every pair of nodes to include the possibility of jumping to a random vertex of the graph:

$$M' = cM + (1 - c) \times [\frac{1}{N}]_{N \times N}$$

A single iteration of the PageRank algorithm is

$$M' \times Rank = cM \times Rank + (1 - c) \times [\frac{1}{N}]_{N \times 1}$$

We implement the external memory algorithm proposed by Haveliwala [8]. The algorithm uses a list of successors *Links*, and two arrays *Source* and *Dest*

that store the vector *Rank* at iteration i and $i + 1$. The computation proceeds until either the error $r = |Source - Dest|$ drops below a fixed value τ or the number of iterations exceed a prescribed value.

Arrays *Source* and *Dest* are partitioned and stored into $\beta = \lceil N/B \rceil$ blocks, each holding the information on B vertices. *Links* is also partitioned into β blocks, where $Links_l$, $l = 0, \dots, \beta - 1$, contains for every vertex of the graph only those successors directed to vertices in block l , i.e. in the range $[lB, (l+1)B - 1]$. We bring to main memory one block of *Dest* per time. Say we have the i th block of *Dest* in main memory. To compute the new PageRank values for all the nodes of the i th block we read, in a streaming fashion, both array *Source* and $Links_i$. From array *Source* we read previous Pagerank values, while from $Links_i$ we have the list of successors (and the out-degree) for each node of the graph to vertices of block i , and these are, from the above Pagerank formula, exactly all the information required.

The main memory occupation is limited to one float for each node in the block, and, in our experiments, 256MB allowed us to keep the whole *Dest* in memory for a 50M vertices graph. Only a small buffer area is required to store *Source* and *Links*, since they are read in a streaming fashion. The time performance of the execution of the algorithm on our synthetic benchmark is shown in Figure 5.

4.3 Strongly Connected Components

It is a well-known fact that SCCs can be computed in linear time by two rounds of depth-first search (DFS). Unfortunately, so far there are no worst-case efficient external-memory algorithms to compute DFS trees for general directed graphs. We therefore apply a recently proposed heuristic for semi-external DFS [17]. It maintains a tentative forest which is modified by I/O-efficiently scanning non-tree edges so as to reduce the number of cross edges. However, this idea does not easily lead to a good algorithm: algorithms of this kind may continue to consider all non-tree edges without making (much) progress. The heuristic overcomes these problems to a large extent by:

- initially constructing a forest with a close to minimal number of trees;
- only replacing an edge in the tentative forest if necessary;
- rearranging the branches of the tentative forest, so that it grows deep faster (as a consequence, from among the many correct DFS forests, the heuristic finds a relatively deep one);
- after considering all edges once, determining as many nodes as possible that have reached their final position in the forest and reducing the set of graph and tree edges accordingly.

The used version of the program accesses at most three integer arrays of size N at the same time plus three boolean arrays. With four bytes per integer and one bit for each boolean, this means that the program has an internal memory requirement of $12.375 \cdot N$ bytes. The standard DFS needs to store $16 \cdot$

avg – degree $\cdot N$ bytes or less if one does not store both endpoints for every edge. Therefore, under memory limitations, standard DFS starts paging at a point when the semi-external approach still performs fine. Figure 6 shows the time performance of the algorithm when applied to graphs generated according to the EN and the Copying model.

4.4 Algorithms for Generating Massive Webgraphs

In this section we present algorithms to generate massive Webgraphs. We consider the Evolving Network model and the Copying model. When generating a graph according to a specific model, we fix in advance the number of nodes N of the simulation. The outcome of the process is a graph stored in secondary memory as list of successors.

Evolving Network model. For the EN model we need to generate the end-point of an edge with probability proportional to the in-degree of a vertex. The straightforward approach is to keep in main memory a N -element array $i[]$ where we store the in-degree for each generated node, so that $i[k] = indegree(v_k) + 1$ (the plus 1 is necessary to give to every vertex an initial non-zero probability to be chosen as end-point). We denote by g the number of vertices generated so far and by I the total in-degree of the vertices $v_1 \dots v_g$ plus g , i.e. $I = \sum_{j=1}^g i[j]$. We randomly (and uniformly) generate a number r in the interval $(1 \dots I)$; then, we search for the smallest integer k such that $r \leq \sum_{j=1}^k i[j]$. For massive graphs, this approach has two main drawbacks: i.) We need to keep in main memory the whole in-degree array to speed up operations; ii.) We need to quickly identify the integer k .

To overcome both problems we partition the set of vertices in $\sqrt{N}$ blocks. Every entry of a $\sqrt{N}$ -element array S contains the sum of the $i[]$ values of a block, i.e. $S[l]$ contains the sum of the elements in the range $i[l \lceil \sqrt{N} \rceil + 1] \dots i[(l+1) \cdot \lceil \sqrt{N} \rceil]$. To identify in which block the end-point of an edge is, we need to compute the smallest k' such that $r \leq \sum_{j=1}^{k'} S[j]$.

The algorithm works by alternating the following 2 phases:

Phase I. We store in main memory tuples corresponding to pending edges, i.e. edges that have been decided but not yet stored. Tuple $t = \langle g, k', r - \sum_{j=1}^{k'-1} S[j] \rangle$ associated with vertex g , maintains the block number k' and the relative position of the endpoint within the block. We also group together the tuples referring to a specific block. We switch to phase II when a sufficiently large number of tuples has been generated.

Phase II. In this phase we generate the edges and we update the information on disk. This is done by considering, in order, all the tuples that refer to a single block when this is moved to main memory. For every tuple, we find the pointed node and we update the information stored in $i[]$. The list of successors is also stored as the graph is generated.

In the real implementation we use multiple levels of blocks, instead of only one, in order to speed up the process of finding the endpoint of an edge. An

alternative is the use of additional data structures to speed up the process of identifying the position of the node inside the block.

Copying model. The Copying model is parameterized with a copying factor α . Every new vertex u inserted in the graph by the Copying model is connected with d edges to previously existing vertices. A random prototype vertex p is also selected. The endpoint of the l th outgoing edge of vertex u , $l = 1, \dots, d$, is either copied with probability α from the endpoint of the l th outgoing link of vertex p , or chosen uniformly at random among the existing nodes with probability $1 - \alpha$.

A natural strategy would be to generate the graph with a batch process that, alternately, i) generates edges and writes them to disk and ii) reads from disk the edges that need to be “copied”. This clearly requires an access to disk for every newly generated vertex.

In the following we present an I/O optimal algorithm that does not need to access the disk to obtain the list of successors of the prototype vertex. We generate for every node $1 + 2 \cdot d$ random integers: one for the choice of the prototype vertex, d for the endpoints chosen at random, and d for the values of α drawn for the d edges. We store the seed of the random number generator at fixed steps, say every x generated nodes.

When we need to copy an edge from a prototype vertex p , we step back to the last time when the seed has been saved before vertex p has been generated, and let the computation progress until the outgoing edges of p are recomputed; for an appropriate choice of x , this sequence of computations is still faster than accessing the disk. Observe that p might also have copied some of its edges. In this case we recursively refer to the prototype vertex of p . We store the generated edges in a memory buffer and write it to disk when complete.

5 Conclusions

In this work we have presented algorithms and experiments for the Webgraph. We plan to carry on these experiments on more recent crawls of the Webgraph in order to assess the temporal evolution of its topological properties. We will also try to get access to the Alexa sample [11] and execute on it our algorithm for disjoint bipartite cliques.

Acknowledgments. We are very thankful to the WebBase project at Stanford and in particular Gary Wesley for their great cooperation. We also thank James Abello, Guido Caldarelli, Paolo De Los Rios, Camil Demetrescu and Alessandro Vespignani for several helpful discussions. We also thanks the anonymous referees for many valuable suggestions.

References

1. R. Albert, H. Jeong, and A.L. Barabasi. *Nature*, (401):130, 1999.
2. O. Riordan B. Bollobas. Robustness and ulnerability of scale-free random graphs. *Internet Mathematics*, 1(1):1–35, 2003.
3. A.L. Barabasi and A. Albert. Emergence of scaling in random networks. *Science*, (286):509, 1999.
4. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
5. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, S. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *Proceedings of the 9th WWW conference*, 2000.
6. C. Cooper and A. Frieze. A general model of undirected web graphs. In *Proc. of the 9th Annual European Symposium on Algorithms(ESA)*.
7. P. Erdős and Renyi R. *Publ. Math. Inst. Hung. Acad. Sci*, 5, 1960.
8. T. H. Haveliwala. Efficient computation of pagerank. Technical report, Stanford University, 1999.
9. J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1997.
10. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proc. of 41st FOCS*, pages 57–65, 2000.
11. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber communities. In *Proc. of the 8th WWW Conference*, pages 403–416, 1999.
12. L. Laura, S. Leonardi, G. Caldarelli, and P. De Los Rios. A multi-layer model for the webgraph. In *On-line proceedings of the 2nd International Workshop on Web Dynamics.*, 2002.
13. L. Laura, S. Leonardi, and S. Millozzi. A software library for generating and measuring massive webgraphs. Technical Report 05-03, DIS - University of Rome La Sapienza, 2003.
14. M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2), 2003.
15. G. Pandurangan, P. Raghavan, and E. Upfal. Using pagerank to characterize web structure. In Springer-Verlag, editor, *Proc. of the 8th Annual International Conference on Combinatorics and Computing (COCOON)*, LNCS 2387, pages 330–339, 2002.
16. D.M. Pennock, G.W. Flake, S. Lawrence, E.J. Glover, and C.L. Giles. Winners don't take all: Characterizing the competition for links on the web. *Proc. of the National Academy of Sciences*, 99(8):5207–5211, April 2002.
17. J.F. Sibeyn, J. Abello, and U. Meyer. Heuristics for semi-external depth first search on directed graphs. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures (SPAA)*, pages 282–292, 2002.
18. J. Vitter. External memory algorithms. In *Proceedings of the 6th Annual European Symposium on Algorithms*, volume 1461 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 1998.
19. The stanford webbase project.
<http://www-diglib.stanford.edu/~testbed/doc2/WebBase/>.

Finding Short Integral Cycle Bases for Cyclic Timetabling*

Christian Liebchen

TU Berlin, Institut für Mathematik, Sekr. MA 6-1
Straße des 17. Juni 136, D-10623 Berlin, Germany
liebchen@math.tu-berlin.de

Abstract. Cyclic timetabling for public transportation companies is usually modeled by the periodic event scheduling problem. To obtain a mixed-integer programming formulation, artificial integer variables have to be introduced. There are many ways to define these integer variables.

We show that the minimal number of integer variables required to encode an instance is achieved by introducing an integer variable for each element of some integral cycle basis of the directed graph $D = (V, A)$ defining the periodic event scheduling problem. Here, integral means that every oriented cycle can be expressed as an *integer* linear combination.

The solution times for the originating application vary extremely with different integral cycle bases. Our computational studies show that the width of integral cycle bases is a good empirical measure for the solution time of the MIP. Integral cycle bases permit a much wider choice than the standard approach, in which integer variables are associated with the co-tree arcs of some spanning tree. To formulate better solvable integer programs, we present algorithms that construct good integral cycle bases. To that end, we investigate subsets and supersets of the set of integral cycle bases. This gives rise to both, a compact classification of directed cycle bases and notable reductions of running times for cyclic timetabling.

1 Introduction and Scope

Cycle bases play an important role in various applications. Recent investigations covering perception in chemical structures ([8]) and the design and analysis of electric networks ([3]). Cyclic timetabling shares with these applications that the construction of a good cycle basis is an important preprocessing step to improve solution methods for real world problems.

Since the pioneering work of Serafini and Ukovich[23], the construction of periodic timetables for public transportation companies, or cyclic timetabling for short, is usually modeled as a periodic event scheduling problem (PESP). For an exhaustive presentation of practical requirements that the PESP is able to meet, we refer to Krista[12]. The feasibility problem has been shown to be $\mathcal{NP}$ -complete, by reductions from Hamiltonian Cycle ([23] and [18]) or Coloring ([20]). The minimization problem with a linear objective has been shown to be $\mathcal{NP}$ -hard by a reduction from Linear Ordering ([16]). We want to solve PESP instances by using the mixed integer solver of CPLEX[®][5].

* Supported by the DFG Research Center “Mathematics for key technologies” in Berlin

Related Work. The performance of implicit enumeration algorithms for mixed integer programming can be improved by reducing the number of integer variables. Already Serafini and Ukovich detected that there is no need to introduce an integer variable for every arc of the directed constraint graph. Rather, one can restrict the integer variables to those that correspond to the co-tree arcs of a spanning tree. These arcs can be interpreted to be the representatives of a strictly fundamental cycle basis.

Nachtigall[17] profited from the spanning tree approach when switching to a tension-based problem formulation. Notice that our results on integral cycle bases apply to that tension-perspective as well. Odijk[20] provided box constraints for the remaining integer variables. Hereby, it becomes possible to quantify the difference between cycle bases. But the implied objective function for finding a short integral cycle basis is bulky. De Pina[21] observed that a cycle basis that minimizes a much simpler function also minimizes our original objective. What remains to solve is a variant of the minimal cycle basis problem.

Contribution and Scope. We show that the width of a cycle basis is highly correlated with the solution time of the MIP solver. Thus, it serves as a good empirical measure for the run time and provides a way to speed up the solver by choosing a good basis.

Hence, in order to supply MIP solvers with promising problem formulations, we want to compute short directed cycle bases which are suitable for expressing PESP instances. But there is a certain dilemma when analyzing the two most popular types of directed cycle bases: On the one hand, there are directed cycle bases that induce undirected cycle bases. For these, we can minimize a linear objective function efficiently (Horton[11]). But, contrary to a claim of de Pina[21], undirected cycle bases unfortunately are *not* applicable to cyclic timetabling in general – we give a counter-example. On the other hand, strictly fundamental cycle bases form a feasible choice. But for them, minimization is $\mathcal{NP}$ -hard (Deo et al.[7]).

To cope with this dilemma, we investigate if there is a class of cycle bases lying in *between* general undirected cycle bases and strictly fundamental cycle bases, hopefully combining both, good algorithmic behavior and the potential to express PESP instances. To that end, we will present a compact classification of directed cycle bases. Efficient characterizations will be based on properties of the corresponding cycle matrices, e.g. its determinant, which we establish to be well-defined. This allows a natural definition of the *determinant of a directed cycle basis*.

An important special class are integral cycle bases. They are the most general structure when limiting a PESP instance to $|A| - |V| + 1$ integer variables. But the complexity of minimizing a linear objective over the integral cycle bases is unknown to the author.

The computational results provided in Section 6 show the enormous benefit of generalizing the spanning tree approach to integral cycle bases for the originating application of cyclic timetabling. These results point out the need of deeper insights into integral cycle bases and related structures. Some open problems are stated at the end.

2 Periodic Scheduling and Short Cycle Bases

An instance of the Periodic Event Scheduling Problem (PESP) consists of a directed constraint graph $D = (V, A, \ell, u)$, where ℓ and u are vectors of lower and upper time bounds for the arcs, together with a period time T of the transportation network. A solution of

a PESP instance is a node potential $\pi : V \rightarrow [0, T)$ —which is a time vector for the periodically recurring departure/arrival events within the public transportation network—fulfilling periodic constraints of the form $(\pi_j - \pi_i - \ell_{ij}) \bmod T \leq u_{ij} - \ell_{ij}$. We reformulate the mod operator by introducing artificial integer variables p_{ij} ,

$$\ell_{ij} \leq \pi_j - \pi_i + p_{ij}T \leq u_{ij}, (i, j) \in A. \quad (1)$$

Our computational results will show that the running times of a mixed-integer solver on instances of cyclic timetabling correlate with the volume of the polytope spanned by box constraints provided for the integer variables. Formulation (1) permits three values $p_a \in \{0, 1, 2\}$ for $a \in A$ in general,¹ even with scaling to $0 \leq \ell_{ij} < T$.

Serafini and Ukovich observed that the above problem formulation may be simplified by eliminating $|V| - 1$ integer variables that correspond to the arcs a of some spanning tree H , when relaxing π to be some real vector. Formally, we just fix $p_a := 0$ for $a \in H$. Then, in general, the remaining integer variables may take more than three values. For example, think of the directed cycle on n arcs, with $\ell \equiv 0$ and $u \equiv T - \frac{1}{n}$, as constraint graph. With $\pi = \mathbf{0}$, the integer variable of every arc will be zero. But $\pi_i = (i - 1) \cdot (T - \frac{1}{n})$, $i = 1, \dots, n$ would be a feasible solution as well, implying $p_{n1} = n - 1$ for the only integer variable that we did not fix to zero. Fortunately, Theorem 1 provides box constraints for the remaining integer variables.

Theorem 1 (Odijk[20]). *A PESP instance defined by the constraint graph $D = (V, A, \ell, u)$ and a period time T is feasible if and only if there exists an integer vector $p \in \mathbb{Z}^{|A|}$ satisfying the cycle inequalities*

$$a_C \leq \sum_{a \in C^+} p_a - \sum_{a \in C^-} p_a \leq b_C, \quad (2)$$

for all (simple) cycles $C \in G$, where a_C and b_C are defined by

$$a_C = \left\lceil \frac{1}{T} \left(\sum_{a \in C^+} \ell_a - \sum_{a \in C^-} u_a \right) \right\rceil, \quad b_C = \left\lfloor \frac{1}{T} \left(\sum_{a \in C^+} u_a - \sum_{a \in C^-} \ell_a \right) \right\rfloor, \quad (3)$$

and C^+ and C^- denote the sets of arcs that, for a fixed orientation of the cycle, are traversed forwardly resp. backwardly.

For any co-tree arc a , the box constraints for p_a can be derived by applying the cycle inequalities (2) to the unique oriented cycle in $H \cup \{a\}$.

Directed Cycle Bases and Undirected Cycle Bases. Let $D = (V, A)$ denote a connected directed graph. An *oriented cycle* C of D consists of *forward arcs* C^+ and *backward arcs* C^- , such that $C = C^+ \dot{\cup} C^-$ and reorienting all arcs in C^- results in a directed cycle. A *directed cycle basis* of D is a set of oriented cycles $C_1, \dots, C_k$ with incidence vectors $\gamma_i \in \{-1, 0, 1\}^{|A|}$ that permit a unique linear combination of

¹ For $T = 10$, $\ell_{ij} = 9$, and $u_{ij} = 11$, $\pi_j = 9$ and $\pi_i = 0$ yield $p_{ij} = 0$; $p_{ij} = 2$ is achieved by $\pi_j = 0$ and $\pi_i = 9$.

the incidence vector of any (oriented) cycle of D , where k denotes the cyclomatic number $k = |A| - |V| + 1$ of D . Arithmetic is performed over the field $\mathbb{Q}$.

For a directed graph D , we obtain the *underlying undirected graph* G by removing the directions from the arcs. A *cycle basis of an undirected graph* $G = (V, E)$ is a set of undirected cycles $C_1, \dots, C_k$ with incidence vectors $\phi_i \in \{0, 1\}^{|E|}$, that again permit to combine any cycle of G . Here, arithmetic is over the field $\text{GF}(2)$. A set of directed cycles $C_1, \dots, C_k$ *projects onto an undirected cycle basis*, if by removing the orientations of the cycles, we obtain a cycle basis for the underlying undirected graph G .

Lemma 1. *Let $\mathcal{C} = \{C_1, \dots, C_k\}$ be a set of oriented cycles in a directed graph D . If $\mathcal{C}$ projects onto an undirected cycle basis, then $\mathcal{C}$ is a directed cycle basis.*

This can easily be verified by considering the mod 2 projection of $\mathcal{C}$, cf. Liebchen and Peeters[15]. But the converse is not true, as can be seen through an example defined on K_6 , with edges oriented arbitrarily ([15]).

Objective Function for Short Cycle Bases. Considering the co-tree arcs in the spanning tree approach as representatives of the elements of a directed cycle basis enables us to formalize the desired property of cycle bases that we need to construct a promising MIP formulation for cyclic timetabling instances.

Definition 1 (Width of a Cycle Basis). *Let $\mathcal{C} = \{C_1, \dots, C_k\}$ be a directed cycle basis of a constraint graph $D = (V, A, \ell, u)$. Let T be a fixed period time. Then, for a_{C_i} and b_{C_i} as defined in (3), we define the width of $\mathcal{C}$ by $W(\mathcal{C}) := \prod_{i=1}^k (b_{C_i} - a_{C_i} + 1)$.*

The width is our empirical measure for the estimated running time of the MIP solver on instances of the originating application. Hence, for the spanning tree approach, we should construct a spanning tree whose cycle basis minimizes the width function. Especially, if many constraints have small *span* $d_a := u_a - \ell_a$, the width will be much smaller than the general bound $3^{|A|}$, which we deduced from the initial formulation (1) of the PESp. To deal with the product and the rounding operation for computing a_{C_i} and b_{C_i} , we consider a slight relaxation of the width:

$$W(\mathcal{C}) \leq \prod_{i=1}^k \left[\frac{1}{T} \sum_{a \in C_i} d_a \right]. \quad (4)$$

De Pina[21] proved that an undirected cycle basis that minimizes the *linearized objective* $\sum_{i=1}^k \sum_{a \in C_i} d_a$ also minimizes the right-hand-side in (4). But there are pathological examples in which a minimal cycle basis for the linearized objective does not minimize the initial width function, see Liebchen and Peeters[15].

Applying the above linearization to spanning trees yields the problem of finding a minimal strictly fundamental cycle basis. But two decades ago, Deo et al.[7] showed this problem to be $\mathcal{NP}$ -hard. Recently, Amaldi[1] established MAX-SNP-hardness.

General Cycle Bases are Misleading. De Pina[21] keeps an integer variable in the PESp only for the cycles of some undirected cycle bases. Consequently, he could exploit

Horton's[11] $\mathcal{O}(m^3n)$ -algorithm² for constructing a minimal cycle basis subject to the linearized objective, in order to find a cycle basis which is likely to have a small width.

In more detail, for a directed cycle basis $\mathcal{C}$, define the *cycle matrix* Γ to be its arc-cycle-incidence matrix. He claimed that the solution spaces stay the same, in particular

$$\{p \in \mathbb{Z}^m \mid p \text{ allows a PESP solution}\} \stackrel{?}{\subseteq} \{\Gamma q \mid q \in \mathbb{Z}^{\mathcal{C}}, q \text{ satisfies (2) on } \mathcal{C}\}. \quad (5)$$

We show that, in general, inclusion (5) does *not* hold. Hartvigsen and Zemel[10] provided a cycle basis $\mathcal{C}$ for their graph M_1 , cf. Figure 1. For our example, we assume

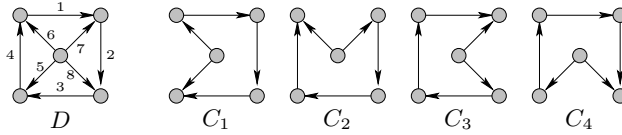


Fig. 1. Cycle basis $\mathcal{C} = \{C_1, \dots, C_4\}$ for which de Pina's approach fails

that the PESP constraints of D allow only the first unit vector e_1 for p in any solution and choose the spanning tree H with $p|_H = 0$ to be the star tree rooted at the center node.

For $\mathcal{C}$, the transpose of the cycle matrix Γ and the inverse matrix of the submatrix Γ' , which is Γ restricted to the rows that correspond to $A \setminus H$, are

$$\Gamma^t = \begin{bmatrix} 1 & 1 & 1 & 0 & -1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & -1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & -1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & -1 \end{bmatrix} \quad \text{and} \quad (\Gamma')^{-1} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 & -2 \\ -2 & 1 & 1 & 1 \\ 1 & -2 & 1 & 1 \\ 1 & 1 & -2 & 1 \end{bmatrix}.$$

The unique inverse image of $p = e_1$ is $q = (\Gamma')^{-1}p|_{A \setminus H} \notin \mathbb{Z}^k$. Thus, the only feasible solution will not be found when working on $\mathbb{Z}^{\mathcal{C}}$. In the following section we will establish that the crux in this example is the fact that there is a regular $k \times k$ submatrix of the cycle matrix whose determinant has an absolute value different from one.

Thus, key information is lost, when only integer linear combinations of the cycles of some arbitrary cycle basis are considered. To summarize, our dilemma is: Cycle bases over which minimization is easy do not fit our purpose. But minimization over cycle bases that are suitable to formulate instances of cyclic timetabling, becomes $\mathcal{NP}$ -hard.

3 Matrix-Classification of Directed Cycle Bases

In order to develop algorithms that construct short cycle bases which we may use for expressing instances of cyclic timetabling, we want to identify an appropriate class of

² Golynski and Horton[9] adapted it to $\mathcal{O}(m^s n)$, with s being the exponent of fast matrix multiplication. By a substantially different approach, de Pina[21] achieved a $\mathcal{O}(m^3 + mn^2 \log n)$ -algorithm for the same problem.

cycle bases. Fortunately, there is indeed some space left between directed cycle bases that project onto undirected ones, and cycle bases which stem from spanning trees. As our classification of this space in between will be based on properties of cycle matrices, we start by giving two algebraic lemmata.

Lemma 2. *Consider a connected digraph D , with a directed cycle basis $\mathcal{C}$ and the corresponding $m \times k$ cycle matrix Γ . A subset of k rows Γ' of Γ is maximal linearly independent, if and only if they correspond to arcs which form the co-tree arcs of a tree.*

Proof. To prove sufficiency, consider a spanning tree H of D , and $\{a_1, \dots, a_k\}$ to become co-tree arcs. Consider the cycle matrix Φ with the incidence vector of the unique cycle in $H \cup \{a_i\}$ in column i . As $\mathcal{C}$ is a directed cycle basis, there is a unique matrix $B \in \mathbb{Q}^{k \times k}$ for combining the cycles of Φ , i.e. $\Gamma B = \Phi$. By construction, the restriction of Φ to the co-tree arcs of H is just the identity matrix. Hence, B is the inverse matrix of Γ' .

Conversely, if the arcs that correspond to the $n - 1$ rows which are *not* in Γ' contain a cycle C , take its incidence vector γ_C . As $\mathcal{C}$ is a directed cycle basis, we have a unique solution $x_C \neq \mathbf{0}$ to the system $\Gamma x = \gamma_C$. Removing $n - 1$ rows that contain C cause x_C to become a non-trivial linear combination of the zero vector, proving Γ' to be singular.

Lemma 3. *Let Γ be the $m \times k$ cycle matrix of some directed cycle basis $\mathcal{C}$. Let A_1 and A_2 be two regular $k \times k$ submatrices of Γ . Then we have $\det A_1 = \pm \det A_2$.*

Proof. By Lemma 2, the k rows of A_1 are the co-tree arcs $a_1, \dots, a_k$ of some spanning tree H . Again, consider the cycle matrix Φ with the incidence vector of the unique cycle in $H \cup \{a_i\}$ in column i . We know that Φ is totally unimodular (Schrijver[22]), and we have $\Phi A_1 = \Gamma$, cf. Berge[2]. Considering only the rows of A_2 , we obtain $\Phi' A_1 = A_2$. As $\det \Phi' = \pm 1$, and as the det-function is distributive, we get $\det A_1 = \pm \det A_2$.

Definition 2 (Determinant of a Directed Cycle Basis). *For a directed cycle basis $\mathcal{C}$ with $m \times k$ cycle matrix Γ and regular $k \times k$ submatrix Γ' , the determinant of $\mathcal{C}$ is*

$$\det \mathcal{C} := |\det \Gamma'|.$$

We first investigate how this determinant behaves for general directed cycle bases, as well as for those who project onto undirected cycle bases.

Corollary 1. *The determinants of directed cycle bases are positive integers.*

Theorem 2. *A directed cycle basis $\mathcal{C}$ projects onto a cycle basis for the underlying undirected graph, if and only if $\det \mathcal{C}$ is odd.*

Due to space limitations, we omit a formal proof and just indicate that taking the mod 2 projection after every step of the Laplace expansion for the determinant of an integer matrix maintains oddness simultaneously over both, $\mathbb{Q}$ and $\text{GF}(2)$.

The following definition introduces the largest class of cycle bases from which we may select elements to give compact formulations for instances of the PESP.

Definition 3 (Integral Cycle Basis). *Let $\mathcal{C} = \{C_1, \dots, C_k\}$ be cycles of a digraph D , where k is the cyclomatic number $k = |A| - |V| + 1$. If, for every cycle C in D , we can find $\lambda_1, \dots, \lambda_k \in \mathbb{Z}$ such that $C = \sum_{i=1}^k \lambda_i C_i$, then $\mathcal{C}$ is an integral cycle basis.*

Theorem 3 (Liebchen and Peeters[15]). *A directed cycle basis $\mathcal{C}$ is integral, if and only if $\det \mathcal{C} = 1$.*

By definition, for every pair of a strictly fundamental cycle basis and an integral cycle basis with cycle matrices Γ and Φ , respectively, there are unimodular matrices B_1 and B_2 with $\Gamma B_1 = \Phi$ and $\Phi B_2 = \Gamma$. Thus, integral cycle bases immediately inherit the capabilities of strictly fundamental cycle bases for expressing instances of cyclic timetabling. Moreover, the example in Figure 1 illustrates that, among the classes we consider in this paper, integral cycle bases are the most general structure for keeping such integer transformations. Hence, they are the most general class of cycle bases allowing to express instances of the periodic event scheduling problem.

Corollary 2. *Every integral cycle basis projects onto an undirected cycle basis.*

The cycle basis in Figure 1 already provided an example of a directed cycle basis that is not integral, but projects onto an undirected cycle basis.

Theorem 3 provides an efficient criterion for recognizing integral cycle bases. But this does not immediately induce an (efficient) algorithm for constructing a directed cycle basis being minimal among the integral cycle bases. Interpreting integral cycle bases in terms of lattices (Liebchen and Peeters[15]) might allow to apply methods for lattice basis reduction, such as the prominent L^3 [13] and Lovász-Scarf algorithms. But notice that our objective function has to be adapted carefully in that case.

4 Special Classes of Integral Cycle Bases

There are two important special subclasses of integral cycle bases. Both give rise to good heuristics for minimizing the linearized width function. We follow the notation of Whitney[24], where he introduced the concept of matroids.

Definition 4 ((Strictly) Fundamental Cycle Basis). *Let $\mathcal{C} = \{C_1, \dots, C_k\}$ be a directed cycle basis. If for some, resp. any, permutation σ , we have*

$$\forall i = 2, \dots, k : C_{\sigma(i)} \setminus (C_{\sigma(1)} \cup \dots \cup C_{\sigma(i-1)}) \neq \emptyset,$$

then $\mathcal{C}$ is called a fundamental resp. strictly fundamental cycle basis.

The following lemma gives a more popular notion of strictly fundamental cycle bases.

Lemma 4. *The following properties of a directed cycle basis $\mathcal{C}$ for a connected digraph D are equivalent:*

1. $\mathcal{C}$ is strictly fundamental.
2. The elements of $\mathcal{C}$ are induced by the chords of some spanning tree.
3. There are at least k arcs that are part of exactly one cycle of $\mathcal{C}$.

We leave the simple proof to the reader.

Hartvigsen and Zemel[10] gave a forbidden minor characterization of graphs in which every cycle basis is fundamental. Moreover, if $\mathcal{C}$ is a fundamental cycle basis such that $\sigma = \text{id}$ complies with the definition, then the first k rows of its arc-cycle incidence matrix Γ constitute an upper triangular matrix with diagonal elements in $\{-1, +1\}$. As an immediate consequence of Theorem 3, we get

Corollary 3. *Fundamental cycle bases are integral cycle bases.*

The converse is not true, as can be seen in a node-minimal example on K_8 , which is due to Liebchen and Peeters[15]. Champetier[4] provides a graph on 17 nodes having a unique minimal cycle basis which is integral but not fundamental. The graph is not planar, as for planar graphs Leydold and Stadler[14] established the simple fact that every minimal cycle basis is fundamental. To complete our discussion, we mention that a directed version of K_5 is a node-minimal graph having a minimal cycle basis which is fundamental, but only in the generalized sense.

The Venn-diagram in Figure 2 summarizes the relationship between the four major subclasses of directed cycle bases.

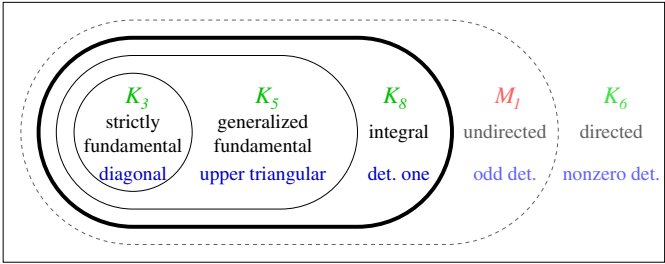


Fig. 2. Map of directed cycle bases

5 Algorithms

A first approach for constructing short integral cycle bases is to run one of the algorithms that construct a minimal undirected cycle basis. By orienting both edges and cycles arbitrarily, the determinant of the resulting directed cycle basis can be tested for having value ± 1 . Notice that reversing an arc's or cycle's direction would translate into multiplying a row or column with minus one, which is of no effect for the determinant of a cycle basis. But if our constructed minimal undirected cycle basis is not integral, it is worthless for us and we have to turn to other algorithms.

Deo et al.[6] introduced two sophisticated algorithms for constructing short strictly fundamental cycle bases: *UV* (unexplored vertices) and *NT* (non-tree edges). But the computational results we are going to present in the next section demonstrate that we can do much better. The key are (generalized) fundamental cycle bases. As the complexity status of constructing a minimal cycle basis among the fundamental cycle bases is unknown to the author, we present several heuristics for constructing short fundamental—thus integral—cycle bases. These are formulated for undirected graphs.

Fundamental Improvements to Spanning Trees. The first algorithm has been proposed by Berger[3]. To a certain extent, the ideas of de Pina[21] were simplified in order to maintain fundamentality. The algorithm is as follows:

1. Set $\mathcal{C} := \emptyset$.
2. Compute some spanning tree H with edges $\{e_{k+1}, \dots, e_m\}$.
3. For $i = 1$ to k do
 - 3.1. For $e_i = \{j, l\}$, find a shortest path P_i between j and l which only uses arcs in $\{e_1, \dots, e_{i-1}, e_{k+1}, \dots, e_m\}$, and set $C_i := e_i \cup P_i$.
 - 3.2. Update $\mathcal{C} := \mathcal{C} \cup C_i$.

Obviously, the above procedure ensures $e_i \in C_i \setminus \{C_1, \dots, C_{i-1}\}$. Hence, $\mathcal{C}$ is a fundamental cycle basis. Although this procedure is rather elementary, Section 6 will point out the notable benefit it achieves even when starting with a rather good strictly fundamental cycle basis, e.g. the ones resulting from the procedures NT or UV. In another context, similar ideas can be found in Nachtigall[19].

Horton's Approximation Algorithm. Horton[11] proposed a fast algorithm for a sub-optimal cycle basis. Below, we show that Horton's heuristic always constructs a fundamental cycle basis for a weighted connected graph G .

1. Set $\mathcal{C} := \emptyset$ and $G' := G$.
2. For $i = 1$ to $n - 1$ do
 - 2.1. Choose a vertex x_i of minimum degree ν in G' .
 - 2.2. Find all shortest paths lengths in $G' \setminus x_i$ between neighbors $x_{i_1}, \dots, x_{i_\nu}$ of x_i .
 - 2.3. Define a new artificial network N_i by
 - 2.3.1. introducing a node s for every edge $\{x_i, x_{i_s}\}$ in G' and
 - 2.3.2. defining the length of the branch $\{s, t\}$ to be the length of a shortest path between x_{i_s} and x_{i_t} in $G' \setminus x_i$.
 - 2.4. Find a minimal spanning tree H_i for N_i .
 - 2.5. Let $C_{i_1}, \dots, C_{i_{\nu-1}}$ be the cycles in G' that correspond to branches of H_i .
 - 2.6. Update $\mathcal{C} := \mathcal{C} \cup \{C_{i_1}, \dots, C_{i_{\nu-1}}\}$ and $G' := G' \setminus x_i$.

Proposition 1. *Horton's approximation algorithm produces a fundamental cycle basis.*

Proof. First, observe that none of the edges $\{x_i, x_{i_s}\}$ can be part of any cycle C_r of a later iteration $r > i$, because at the end of iteration i the vertex x_i is removed from G' . Hence, fundamentality follows by ordering, within each iteration i , the edges and cycles such that $e_{i_j} \in C_{i_j} \setminus (C_{i_1}, \dots, C_{i_{j-1}})$ for all $j = 2, \dots, \nu - 1$. Moreover, every leaf s of H_i encodes an edge $\{x_i, x_{i_s}\}$ that is part of only one cycle. Finally, as H_i is a tree, by recursively removing branches that are incident to a leaf of the remaining tree, we process every branch of the initial tree H_i .

We order the branches $b_1, \dots, b_{\nu-1}$ of H_i according to such an elimination scheme, i.e. for every branch $b_j = \{s_j, t_j\}$, node s_j is a leaf subject to the subtree $H_i \setminus \bigcup_{\ell=1}^{j-1} \{b_\ell\}$. Turning back to the original graph G' , for $j = 1, \dots, \nu - 1$, we define e_{i_j} to correspond to the leaf $s_{\nu-j}$, and C_{i_j} to be modeled by the branch $b_{\nu-j}$. This just complies with the definition of a fundamental cycle basis.

6 Computational Results

The first instance has been made available to us by Deutsche Bahn AG. As proposed in Liebchen and Peeters[16], we want to minimize simultaneously both the number of vehicles required to operate the ten given pairs of hourly served ICE/IC railway lines, and the waiting times faced by passengers along the 40 most important connections. Single tracks and optional additional stopping times of up to five minutes at major stations cause an average span of 75.9% of the period time for the 186 arcs that remain after elimination of redundancies within the initial model with 4104 periodic events.

The second instance models the Berlin Underground. For the eight pairs of directed lines, which are operated every 10 minutes, we consider all of the 144 connections for passengers. Additional stopping time is allowed to insert for 22 stopping activities. Hereby, the 188 arcs after eliminating redundancies have an average span of 69.5% of the period time. From earlier experiments we know that an optimal solution inserts 3.5 minutes of additional stopping time without necessitating an additional vehicle. The weighted average passengers' effective waiting time is less than 1.5 minutes.

For the ICE/IC instance, in Table 1 we start by giving the base ten logarithm of the width of the cycle bases that are constructed by the heuristics proposed in Deo et al.[6] These have been applied for the arcs' weights chosen as unit weights, the span $d_a = u_a - \ell_a$, or the negative of the span $T - d_a$. In addition, minimal spanning trees have been computed for two weight functions. The fundamental improvement heuristic has been applied to each of the resulting strictly fundamental cycle bases, For sake of completeness, the width of a minimal cycle basis subject to the linearized objective is given as well. The heuristic proposed by Horton has not been implemented so far.

Subsequently, we report the behavior of CPLEX[®][5] when faced with the different problem formulations. We use version 8.0 with standard parameters, except for *strong branching* as variable selection strategy and *aggressive cut generation*. The computations have been performed on an AMD Athlon[®] XP 1500+ with 512 MB main memory.

Table 1. Influence of cycle bases on running times for timetabling (hourly served ICE/IC lines)

algorithm weight	global	MST		UV			NT
	minima	span	nspan	unit	span	nspan	unit
initial width	34.3	65.9	88.4	59.7	58.6	61.2	58.5
fund. improve	–	41.0	43.2	42.9	42.2	42.9	42.7
without fundamental improvement							
time (s)	–	14720	>28800	20029	23726	6388	>28800
memory (MB)	–	13	113	29	30	10	48
status	–	–	opt	opt	opt	opt	timelimit
solution	620486	–	667080	–	–	–	629993
fundamental improvement applied							
time (s)	–	807	11985	9305	17963	1103	>28800
memory (MB)	–	1	23	24	30	3	114
status	–	–	opt	opt	opt	opt	timelimit
solution	–	–	–	–	–	–	626051

Due to space limitations, we just summarize that the solution behavior is the same for the instance of the Berlin Underground. The width of a minimal cycle basis is about 10^{39} , and the fundamental improvement reduced the width from values between 10^{62} and 10^{85} down to values ranging from 10^{46} to only 10^{49} . The only computation which exceeded our time limit is again MST nspan without fundamental improvement. Only 19 seconds were necessary to optimize the improved UV nspan formulation.

A key observation is the considerable positive correlation (> 0.44 and > 0.67) between the base ten logarithm of the width of the cycle basis and the running time of the MIP solver. With the exception of only one case, the fundamental improvement either results in a notable speed-up, or enables an instance to be solved to optimality, in case that the time limit of eight hours is reached when not applying the heuristic. Figure 3 provides a detailed insight into the distribution of cycle widths of the basic cycles for the ICE/IC instance before and after the fundamental improvement.

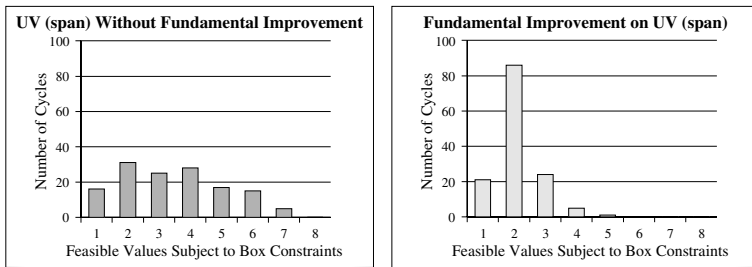


Fig. 3. Shift in distribution of cycle widths due to the fundamental improvements

Since the known valid inequalities, e.g. (2) and Nachtigall[18], heavily depend on the problem formulation, they have not been added in any of the above computations. However, they also provide a major source for improving computation times. For the instance of Deutsche Bahn AG, an optimal solution was obtained after only 66 seconds of CPU time for a formulation refined by 115 additional valid inequalities which were separated in less than 80 seconds.

7 Conclusions

We generalized the standard approach for formulating the cyclic timetabling problem, based on strictly fundamental cycle bases. Integral cycle bases have been established to be the most general class of directed cycle bases that enable the modeling of cyclic timetabling problems. Finally, we presented algorithms that construct short fundamental cycle bases with respect to a reliable empirical measure for estimating the running time of a mixed-integer solver for the originating application.

But some questions remain open. One is the complexity status of minimizing a (linear) objective function over the class of fundamental, or even integral, cycle bases. Another is progress in the area of integer lattices. Finally, it is unknown, whether every graph has a minimal cycle basis that is integral.

Acknowledgments. Franziska Berger, Bob Bixby, Sabine Cornelsen, Berit Johannes, Rolf H. Möhring, Leon Peeters, and of course the anonymous referees contributed in various ways to this paper.

References

1. Amaldi, E. (2003) Personal Communication. Politecnico di Milano, Italy
2. Berge, C. (1962) *The Theory of Graphs and its Applications*. John Wiley & Sons
3. Berger, F. (2002) Minimale Kreisbasen in Graphen. Lecture on the annual meeting of the DMV in Halle, Germany
4. Champetier, C. (1987) On the Null-Homotopy of Graphs. *Discrete Mathematics* **64**, 97–98
5. CPLEX 8.0 (2002) <http://www.ilog.com/products/cplex> ILOG SA, France.
6. Deo, N., Kumar, N., Parsons, J. (1995) Minimum-Length Fundamental-Cycle Set Problem: A New Heuristic and an SIMD Implementation. Technical Report CS-TR-95-04, University of Central Florida, Orlando
7. Deo, N., Prabhu, M., Krishnamoorthy, M.S. (1982) Algorithms for Generating Fundamental Cycles in a Graph. *ACM Transactions on Mathematical Software* **8**, 26–42
8. Gleiss, P. (2001) Short Cycles. Ph.D. Thesis, University of Vienna, Austria
9. Golynski, A., Horton, J.D. (2002) A Polynomial Time Algorithm to Find the Minimum Cycle Basis of a Regular Matroid. In: SWAT 2002, Springer LNCS 2368, edited by M. Penttonen and E. Meineche Schmidt
10. Hartvigsen, D., Zemel, E. (1989) Is Every Cycle Basis Fundamental? *Journal of Graph Theory* **13**, 117–137
11. Horton, J.D. (1987) A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM Journal on Computing* **16**, 358–366
12. Krista, M. (1996) Verfahren zur Fahrplanoptimierung dargestellt am Beispiel der Synchronzeiten (Methods for Timetable Optimization Illustrated by Synchronous Times). Ph.D. Thesis, Technical University Braunschweig, Germany, In German
13. Lenstra, A.K., Lenstra, H.W., Lovász, L. (1982) Factoring polynomials with rational coefficients. *Mathematische Annalen* **261**, 515–534
14. Leydold, J., Stadler, P.F. (1998) Minimal Cycle Bases of Outerplanar Graphs. *The Electronic Journal of Combinatorics* **5**, #16
15. Liebchen, C., Peeters, L. (2002) On Cyclic Timetabling and Cycles in Graphs. Technical Report 761/2002, TU Berlin
16. Liebchen, C., Peeters, L. (2002) Some Practical Aspects of Periodic Timetabling. In: *Operations Research 2001*, Springer, edited by P. Chamonì et al.
17. Nachtigall, K. (1994) A Branch and Cut Approach for Periodic Network Programming. *Hildesheimer Informatik-Berichte* 29
18. Nachtigall, K. (1996) Cutting planes for a polyhedron associated with a periodic network. *DLR Interner Bericht* 17
19. Nachtigall, K. (1996) Periodic network optimization with different arc frequencies. *Discrete Applied Mathematics* **69**, 1–17
20. Odijk, M. (1997) Railway Timetable Generation. Ph.D. Thesis, TU Delft, The Netherlands
21. de Pina, J.C. (1995) Applications of Shortest Path Methods. Ph.D. Thesis, University of Amsterdam, The Netherlands
22. Schrijver, A. (1998) *Theory of Linear and Integer Programming*. Second Edition. Wiley
23. Serafini, P., Ukovich, W. (1989) A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics* **2**, 550–581
24. Whitney, H. (1935) On the Abstract Properties of Linear Dependence. *American Journal of Mathematics* **57**, 509–533

Slack Optimization of Timing-Critical Nets

Matthias Müller-Hannemann and Ute Zimmermann

Research Institute for Discrete Mathematics
Rheinische Friedrich-Wilhelms-Universität Bonn
Lennéstr. 2, 53113 Bonn, Germany
{muellerh,zimmerm}@or.uni-bonn.de

Abstract. The construction of buffered Steiner trees becomes more and more important in the physical design process of modern chips. In this paper we focus on delay optimization of timing-critical buffered Steiner tree instances in the presence of obstacles. As a secondary goal, we are interested in minimizing power consumption.

Since the problem is NP-hard, we first study an efficient method to compute upper bounds on the achievable slack. This leads to the interesting subproblem to find shortest weighted paths under special length restrictions on routing over obstacles. We prove that the latter problem can be solved efficiently by Dijkstra's method. In the main part we describe a new approach for the buffered Steiner tree problem. The core step is an iterative clustering method to build up the tree topology.

We provide a case study for the effectiveness of the proposed method to construct buffered Steiner trees. Our computational experiments on four different chip designs demonstrate that the proposed method yields results which are relatively close to the slack bounds. Moreover, we improve significantly upon a standard industry tool: we simultaneously improve the slack and largely reduce power consumption.

Keywords: Buffered rectilinear Steiner trees, VLSI design, upper bounds, clustering, blockages

1 Introduction and Overview

Steady advances in integrated circuit technology have led to much smaller and faster devices so that interconnect delay becomes the bottleneck in achieving high-performance integrated circuits. Interconnect delay can be reduced by insertion of buffers and inverters¹. On increasingly complex integrated circuits buffer insertion needs to be performed on several thousands of nets. Since buffers and inverters are implemented by transistors, it is impossible to place them over existing macro blocks or other circuits. Thus, such blocks are obstacles for buffer insertion. This paper studies the problem of buffered routing tree construction in the presence of obstacles. We shall focus on delay optimization

¹ A *buffer* (also called *repeater*) is a circuit which logically realizes the identity function $id : \{0, 1\} \mapsto \{0, 1\}, id(x) = x$, whereas an *inverter* realizes logical negation.

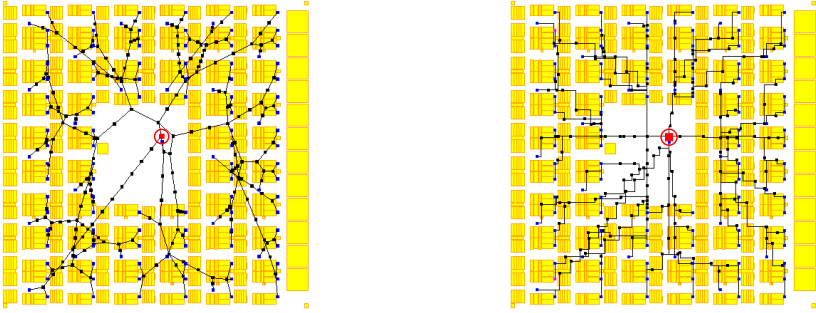


Fig. 1. A typical instance with a barycentring embedding of the topological tree (left) and the corresponding legalized rectilinear embedding (right) of the tree constructed by our code. The source of the net is encircled.

of timing-critical routing tree instances. As a secondary goal, we are interested in minimizing power consumption.

Problem definition. We consider the problem to connect a source with a set of sinks by a buffered Steiner tree such that we can send a signal from the source to the sinks. A (*signal*) *net* $N = \{s, t_1, t_2, \dots, t_k\}$ is a set of $k + 1$ terminals, where s is the *source* and the remaining terminals t_i are *sinks*. The source and the sinks correspond to pins of circuits. Each sink has a *required arrival time* $rat(t_i)$ for the signal, an *input capacitance* $incap(t_i)$ and a *polarity constraint* $pol(t_i) \in \{0, 1\}$. The constraint $pol(t_i) = 1$ requires the inversion of the signal from s to t_i , whereas $pol(t_i) = 0$ prohibits the signal inversion. We are given a *library* B of inverter and buffer types with different characteristics. Roughly speaking, a larger and therefore stronger inverter or buffer type has a larger input capacitance but causes a smaller delay.

An *edge* is a horizontal or vertical line connecting two points in the plane. A *rectilinear tree* is a connected acyclic collection of edges which intersect only at their endpoints. A *rectilinear Steiner tree* $T = (V, E)$ for a given set of terminals $N \subseteq V$ is a rectilinear tree such that each terminal is an endpoint of some edge in the tree. A *buffered Steiner tree* $T_b = (V_b, E_b)$ for the net N is a rectilinear Steiner tree where $V_b = N \cup I \cup S$. Here I denotes a set of nodes corresponding to buffers and inverters, and S denotes Steiner points of the tree. Given a buffered Steiner tree, we consider the tree as rooted at the source of the net and its edges as directed away from the source. For each node $v \in I$, let $bhc(v)$ (*block hardware code*) be the inverter or buffer type chosen from the given library B . Feasible positions for the placement of buffers and inverters are only those regions which are not blocked by obstacles. Throughout this paper, an *obstacle* is a connected region in the plane bounded by one or more simple rectilinear polygons such that no two polygon edges have an inner point in common (i.e. an obstacle may contain holes). For a given set of obstacles $\mathcal{O}$ we require that the obstacles be disjoint, except for possibly a finite number of

common points. In real world applications, most obstacles are rectangles or of very low complexity, see Fig. 1.

The buffered Steiner tree construction has to respect several side constraints. *Polarity constraints* require that the number of inverters on the unique path from the source s to sink t_i is even if and only if $pol(t_i) = 0$. For an edge $e = (u, v)$ of a Steiner tree with length $\ell(e)$, $r(e) := r_w \cdot \ell(e)$ is the *wire resistance* of e and $c(e) := c_w \cdot \ell(e)$ is the *wire capacitance* of e , with r_w and c_w being unit wire resistance and unit wire capacitance factors, respectively. Every circuit v has a *maximum load capacitance* $maxoutcap(v)$ which it can drive. Its *actual load* is the downstream capacitance of its “visible subtree” $T(v)$. For any node v of a buffered Steiner tree, the *visible subtree* $T(v)$ is the tree induced by v and all those nodes w which can be reached by a directed path from v with the property that all nodes on this path except v and w are Steiner nodes. Each node corresponding to a circuit has an *input capacitance* $incap(v)$ specified in the given library, for all Steiner nodes we define $incap(v) := 0$. Thus, the load of a node v is defined as the capacitance of its visible subtree $outcap(v) := c(T(v)) := \sum_{u \in V(T(v)) \setminus \{v\}} incap(u) + \sum_{e \in E(T(v))} c(e)$, and we have to fulfill the *load constraints* $outcap(v) \leq maxoutcap(v)$ for each circuit v . A buffered Steiner tree is *feasible* if it respects all polarity conditions and satisfies all load constraints.

Let $delay(s, v, T_b)$ be the *delay* of a signal from s to v within a buffered Steiner tree T_b . This delay can be computed as the sum of the delays through circuits and wire delay, on the unique path in T_b from s to v . To compute a delay for a node v which corresponds to a circuit, we assume to have an efficiently computable black-box function $delay(v) := delay(bhc(v), outcap(v))$, which depends on the block hardware code $bhc(v)$ of v and its output capacitance $outcap(v)$. The delay through a Steiner node is zero. We use the Elmore delay model for wire delays. The *Elmore delay* of an edge $e = (u, v)$ is given by $delay(e) := r(e)(c(e)/2 + c(T_v))$. Note that the edge delay depends quadratically on the edge length. The *slack* of a tree T is given by $slack(T_b) := \min_{1 \leq i \leq k} \{rat(t_i) - delay(s, t_i, T_b)\}$. The primary objective which we consider in this paper is to maximize the slack. In particular, if the slack is non-negative, then each signal arrives in time. Among all buffered Steiner trees which achieve a certain slack, we try to minimize power consumption as a secondary objective. The variable part of the power consumption of a buffered Steiner tree can be assumed to be proportional to the tree capacitance. Clearly, maximizing the slack for a buffered Steiner tree instance is NP-hard, as the problem contains several NP-hard special cases, most obviously the NP-hard rectilinear Steiner tree problem [6].

Previous work. Given a fixed Steiner tree and a finite set of possible locations for buffer insertion, dynamic programming can achieve a slack-optimal buffering under the Elmore delay model [13,11]. Hence, the hardness of the buffered Steiner tree problem lies in the construction of a good tree topology. Alpert et al. [2] propose a two-stage algorithm, called *C-tree*, that first clusters sinks

with common characteristics together, builds a Steiner tree for each cluster, and finally builds a Steiner tree connecting the clusters. A weakness of the C-tree approach lies in the decision of how many clusters should be used which appears to be very instance-dependent. Moreover, the second stage to connect the clusters uses a Dijkstra-Prim heuristic [3] and therefore does neither take sink criticality nor sink polarity into account. This approach also ignores blockages.

Tang et al.[12] considered the construction of buffered Steiner trees on a grid in the presence of routing and buffer obstacles by a variant of dynamic programming based on auxiliary graphs and table lookup. Their graph-based algorithm, however, requires space and runtime growing exponentially in the number of sinks. For instances with many sinks, the authors therefore propose a staged approach which first applies clustering and then combines the clusters by the graph-based algorithm in the second phase. A major drawback of this approach is that it only tries to minimize the maximum delay but does not take individual required arrival times at the sinks into account.

Our contribution and overview. In Section 2 we first consider the task to compute upper bounds on the achievable slack for buffered routing tree instances. As a slack bound we propose the minimum slack value obtained from computing slack-optimal paths between all source-sink pairs of an instance. However, even computing a slack-optimal path is a highly non-trivial task. Therefore, instead of computing exact slack-optimal paths we only aim at high-quality approximations of these instances. Our approach for path instances runs in two phases: In the first phase we search for a path which is a suitable compromise between the conflicting goals to minimize path length on the one hand and to avoid obstacles and dense regions on the other hand. In the second phase we use dynamic programming on this path for optimal buffer insertion. For the path obtained from the first phase we have to guarantee that the distance on the path between two feasible positions for buffer insertion becomes nowhere too large. Otherwise there may be no buffering which can meet the load restrictions. This leads to an interesting subproblem (which has not been studied so far): we have to find shortest rectilinear paths under length restrictions for subpaths which run over obstacles. We prove that this problem can efficiently be solved by Dijkstra's algorithm on an auxiliary graph which is a variant of the so-called Hanan grid. We use the slack bounds for two purposes: on the one hand, they provide us with a guarantee for the quality of our solutions. On the other hand they are used to guide our strategy to build the tree topology.

In Section 3 we describe our new approach for the construction of buffered Steiner trees. The basic idea is to use an iterative clustering approach to build up the tree topology. In our approach there is no a priori decision on how many clusters should be used. In Section 4 we present results of a computational study on the solution quality obtained from an implementation of the proposed methods. To this end, we use several thousand test instances generated from four recent chip designs with very different data profiles. Previous computational studies typically considered only small or artificial instances. We are not aware of a study which uses bounds and thereby presents performance guarantees. It turns

out that our heuristic solutions achieve slacks which are relatively close to the upper slack bounds. However, the gap between the upper bound and the achieved slack increases with the number of sinks. Finally, our code is compared with a standard software tool currently used by IBM. We obtain significant improvements over this tool, simultaneously with respect to slack, power consumption and total wire length.

2 Computing Upper Bounds for the Slack

2.1 A Two-Stage Approach

To compute upper bounds for the slack, we consider the special instances which are induced by the source s and a single sink t_i for $i = 1, \dots, k$. Clearly, if we compute a slack-optimal path for each of these special instances we can afterwards take the minimum slack of them as an upper bound. The only problem is that even optimizing path instances is a non-trivial task.

Several authors used an analytical approach to determine the optimal number and placement of buffers for delay minimization on a single path [1,4,9]. Assuming a single buffer or inverter type, using a simplified delay model and completely ignoring blockages, this basically leads to an equidistant placement of buffers, possibly with a shift towards the root or to the sink depending on their relative strengths. Clearly, such simple solutions are very efficient, but experiments show that they tend to be quite inaccurate (if adapted to work for general scenarios). On the other extreme, exact solutions are rather expensive. Zhou et al. [15] proposed to determine the fastest path by a dynamic programming approach on a grid graph where the grid nodes correspond to possible placement positions of inverters and buffers. The running time of this method is $O(|B|^2 n^2 \log(n|B|))$, where n denotes the number of grid nodes and $|B|$ is the number of inverter- and buffer types in the library. However, even after recent advances and speed-ups by Lai & Wong [10] and Huang et al. [8] these approaches are by orders of magnitude too slow in practice.

Therefore, in order to improve the efficiency we use a two-stage approach. In the first stage, we search for a path which avoids obstacles as far as possible. This idea will be made precise in the following subsection. In the second stage, we determine a finite set of legal positions for inverters and buffers on this path. With respect to these positions we determine by dynamic programming an optimal choice of inverter types from the given library for the given positions.

2.2 Shortest Length-Restricted Paths

We introduce length restrictions for those portions of a path P which run over obstacles. Note that the intersection of a path with an obstacle may consist of more than one connected component. Our length restriction applies individually for each connected component. Every obstacle O is weighted with a factor $w_O \geq 1$ (regions not occupied by an obstacle and boundaries of obstacles all have unit

weight). By ∂O we denote the boundary of an obstacle O . For each obstacle $O \in \mathcal{O}$, we are given a parameter $L_O \in \mathbb{R}_0^+$. Now we require for each obstacle $O \in \mathcal{O}$ and for each strictly interior connected component P_O of $(P \cap O) \setminus \partial O$ that the (weighted) length $\ell(P_O)$ of such a component must not be longer than the given length restriction L_O . Note that, by setting $L_O = 0$ for an obstacle, we can model the case that the interior of O must be completely avoided.

Problem 1 (Length-restricted shortest path problem (LRSP)).

Instance: Two points s and t in the plane, a set of (weighted) obstacles $\mathcal{O}$, and length restrictions $L_O \in \mathbb{R}_0^+$ for $O \in \mathcal{O}$.

Task: Find a rectilinear path P of minimum (weighted) length such that for all obstacles $O \in \mathcal{O}$, all connected components P_O of $(P \cap O) \setminus \partial O$ satisfy $\ell(P_O) \leq L_O$.

Given a finite point set S in the plane and a set of obstacles $\mathcal{O}$, the *Hanan grid* [7] is obtained by constructing a vertical and a horizontal line through each point of S and a line through each edge used in the description of the obstacles. In our scenario, we just have $S = \{s, t\}$. It is well-known that the Hanan grid contains a rectilinear shortest path (without length restrictions). In fact, this holds even for several generalizations to minimum rectilinear Steiner trees, see Zachariasen's catalog [14]. Fortunately, we can still guarantee that there is an optimal length-restricted shortest path which uses only Hanan grid edges.

Lemma 1. *Given two terminals s, t , a set of obstacles $\mathcal{O}$ and length restrictions L_O for $O \in \mathcal{O}$, there is an optimal length-restricted (s - t)-path using only Hanan grid edges.*

Lemma 2. *Given a Hanan grid with n nodes, there is a graph G with $O(n)$ nodes and edges which contains only length-feasible s - t -paths (and, in particular, an optimal s - t -path). Such a graph can be constructed in $O(n)$ time. Hence, the length-restricted shortest path problem can be solved in $O(n \log n)$ time by Dijkstra's algorithm.*

Practical considerations. As we have to solve many thousands of shortest path problems, it is crucial to build up the Hanan grid data structure only once. A technical difficulty lies in the handling of the rows and columns corresponding to s and t since these lines change for each query. Thus, we use a linear time preprocessing to dynamically modify the Hanan grid for each query.

By setting the weight for internal obstacle edges to 1.01 we allow only a 1% increase in total wire length. In particular, we thereby avoid all obstacles if there is a shortest unweighted s - t -path which does so. However, to avoid large obstacles, we have to increase the penalty.

3 Construction of Slack-Critical Inverter Trees

3.1 Main Steps of Our Tree Construction

Let us first give a high-level description of our approach. The following subsections will describe the individual steps in more detail. A fundamental step in the construction of a buffered Steiner tree is to determine the tree topology. Let us first assume that we have to deal with an instance where all sinks have roughly the same criticality. (We remark that this is typically the case in the early phase of timing optimization when no meaningful required arrival times are available.) The key idea for constructing a tree topology is to apply an iterative clustering scheme which takes the sinks and source of a net as input. Sinks are greedily clustered together based on their spatial proximity and polarity. Members of the same cluster shall be driven by the same inverter or buffer. This implies that each cluster has to contain only members of the same polarity. Furthermore, due to the load limit for buffers and inverters, cluster sizes have to be restricted. Hence, we “close” a cluster if no further clustering is possible without violating these size limits. In such an event, we insert an inverter for the closed cluster and make all cluster members its children in the tree topology we want to create. The inserted inverter then plays the role of a new sink (with the corresponding opposite polarity) but will appear on a higher level in the hierarchy of the tree.

Of course, the sinks may have very different criticality. Here is the point where we use the upper bounds for the achievable slack of each sink as a measure for its criticality. Namely, we compute these slack bounds and sort the sinks according to increasing criticality. We use this order on the sinks (and also the distribution of the slack bound values) to partition the set of all sinks into “critical sinks” and “non-critical sinks”. (Since there is not always a natural partition into critical and non-critical sinks, we repeat the following steps with up to three different partitions.) Afterwards we apply the clustering heuristic individually for these sink sets and unite the resulting trees which gives us the tree topology for the overall instance. Based on this tree topology the next steps try to find an improved tree embedding, to optimize the gate sizing for the inverter types (choice of block-hardware codes) and finally to reduce the power consumption on subtrees with a positive slack. We summarize the main steps:

1. Compute upper bounds for the achievable slack for each sink.
2. Partition the set of sinks into critical sinks P_1 and non-critical sinks P_2 .
3. Use clustering heuristic, to construct tree topologies for the sink sets P_i , and unite the two trees.
4. Try to improve the embedding.
5. Optimize gate sizing with respect to the slack.
6. Reduce tree capacitance on subtrees with a positive slack.

3.2 An Iterative Clustering Approach

We use the following data structure to implement our clustering. A *cluster* (a) contains a set of circuits (inverters, buffers, the root, or sinks) with corresponding block-hardware codes, (b) has a parity, (c) maintains a bounding box, (d)

has a cluster size, (e) is active or inactive, and (f) has a pointer to some circuit in a parent cluster (if inactive). The *size of a cluster* is an estimation of the capacitance of the induced net. For example, an easy to calculate estimation of the capacitance is simply the sum of the input capacitances of the circuits plus wire capacitance estimated by the bounding box length of its circuits. The size of each cluster is restricted by an *upper size limit*. This upper size limit depends on several parameters like the available library of inverters and buffers, and others. Roughly speaking, it is chosen such that a mid-size inverter from the library can drive the circuits of any cluster. Initially, each node of the net becomes a singleton cluster. Each cluster is active in the beginning, and becomes inactive if it has been closed. The *root cluster* for the source node s plays a special role as it should never be closed. Hence, it is always active and has a circuit-specific, usually smaller upper size limit. The algorithm terminates when all clusters except for the root cluster are inactive. Each inactive cluster has exactly one parent circuit. This parent circuit together with the members of a cluster form a net of the topological tree structure. Our algorithm *Greedy Clustering* works as follows:

Algorithm Greedy Clustering

Input: the root s , a set of sinks $t_1, \dots, t_k$ with parities and coordinates

Output: an inverter tree structure rooted at s

- Step 0: initialize $C_0 = \{s\}$, $C_i = \{t_i\}$ as active clusters with corresponding polarity;
- Step 1: search for a pair C_i, C_j of active clusters with same polarity, such that their union has smallest cluster size;
- Step 2: **if** the combined cluster size of C_i and C_j is smaller than an upper cluster size limit, **then** unite the clusters;
- Step 3: **else** (i. e. there is no suitable pair of clusters) open a new cluster C_k , make one cluster in-active (for example the largest) and child of C_k ; find suitable position for C_k ;
- Step 4: **if** more than one cluster is active, **then** goto Step 1 and iterate;

It remains to explain Steps 2 and 3 in more detail. *Uniting two clusters* requires that both clusters have the same parity. In this case, a new cluster with the same parity replaces both previous clusters. It contains as elements the union of the elements from the two former clusters. The new bounding box and cluster size are determined from the previous values in constant time. *Opening a new cluster* means to select an inverter or buffer type from the library, to determine a (feasible) circuit position on the chip image, and to choose an existing active cluster as a child. The child cluster becomes inactive by this operation. The circuit position is chosen to lie “somewhat closer” to the source than the bounding box of the child cluster. More precisely, assuming that the source lies outside the bounding box, we determine a weighted shortest path from the source s to the nearest point p of the bounding box. Then we go from p on this path towards the source s until load of this circuit is filled up to its limit. If the bounding box of the child cluster already contains the source then we choose the nearest legal position to the source. The initial values of a new cluster are then determined as follows. The status of the cluster is active, its

bounding box is the selected position, its size is the input capacitance of the selected inverter or buffer type, and its parity is opposite to the child cluster's parity if and only if the selected circuit is an inverter. The algorithm terminates because in each iteration we either strictly decrease the number of remaining clusters by one in Step 2, or we replace one previous cluster by a new cluster which lies closer to the source. At termination, all members (but the source) of the remaining root cluster become children of the source s .

3.3 Tree Embedding and Legalization

Next we consider the problem to find a good tree embedding given a fixed tree topology. Of course, the clustering heuristic already yields a feasible, first tree embedding. However, as the clustering heuristic has to determine positions of the interior tree vertices based on partial knowledge of the tree structure (at the decision, we neither know the parent node nor its siblings), this embedding is likely to be far from optimal. Note that we have three partially conflicting goals for a tree embedding: (1) The overall tree length should be as small as possible to minimize power consumption. (2) Source-sink paths should be as short as possible for a small delay. (3) Long edges should be avoided as the wire delay is quadratic in the edge length.

Force-directed tree embedding. Let us ignore blockages for a moment. We use a heuristic inspired by force-directed methods in graph drawing [5]. The idea is to specify attracting forces between adjacent tree vertices proportional to their distance. The force-directed method then determines positions for all movable vertices (i.e. all vertices except the source and the sinks) which correspond to a force equilibrium. It is easy to see that such an equilibrium is attained if each movable vertex is placed to the barycentre of its tree neighbors. Such a barycentric embedding has the nice property to minimize the sum of the squared Euclidian edge lengths. This objective function can be seen as a reasonable compromise between the first and the third goal. If we consider a weighted version where we give higher weights to edges on paths to critical sinks, we can also capture the second goal to a certain extent. Additional forces can be modeled by artificial non-tree edges. For example, it is common practice in placement algorithms to have attractive forces between all pairs of sibling nodes of the tree.

Legalization and blockage avoidance. To legalize the embedding with respect to blockages, a very simple approach is just to move each inverter or buffer to the nearest feasible position. Slightly better is to iteratively select the best position (with respect to the resulting slack) among at most four choices, namely the nearest feasible position with respect to positive and negative coordinate directions. If the repositioning leads to considerably longer paths load violations may occur. However, by inserting additional inverters or buffers on this path, such load violations can be avoided. Both heuristics seem to be somewhat naive, but they work surprisingly well as our experiments indicate. To avoid large blockages or dense placement regions, we reuse our weighted shortest path approach with length restrictions from Section 2.2 to reembed subpaths of the tree.

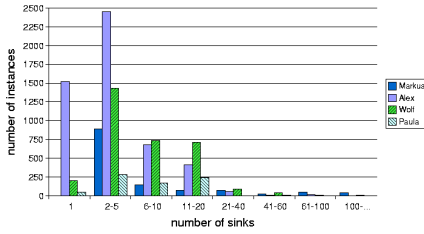


Fig. 2. The number of instances with respect to the number of sinks.

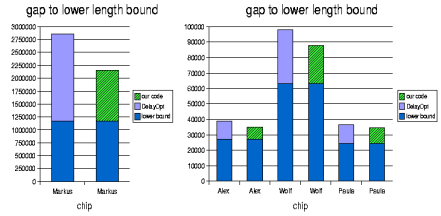


Fig. 3. The length of our buffered Steiner trees in comparison with DelayOpt and the length of a Steiner minimum tree. All lengths are given in measurement units $MU = 10^{-8}m$.

3.4 Gate Sizing and Power Reduction

As mentioned earlier, slack-optimal gate sizing can be achieved by straightforward dynamic programming on a given tree topology. Unfortunately, this is a relatively expensive approach. As an alternative, we use local improvement strategies to optimize the slack of a given tree topology. A sink is called *critical sink* if it has minimum slack among all sinks. The corresponding path from the critical sink to the source is the *critical path*. Clearly, to improve the overall slack we have to reduce the delay along the critical path.

Large wire delays of edges on the critical path which increase a certain threshold may be reduced by inserting additional inverters or buffers. To reduce the delay of circuits on the critical path, we have several possibilities for each circuit on the critical path: (a) We can vary the choice of the inverter or buffer type by exchanging the block hardware code. Note that, in general, a larger type will locally yield a shorter delay, but may negatively influence the delay of its predecessor on the path due to its larger input capacitance. (b) We can try to reduce the load by selecting a smaller block hardware code for a child node which does not belong to the critical path. (c) Another local operation (not available in dynamic programming approaches) for load reduction is to shorten wire capacitance by moving non-critical sibling nodes of critical nodes nearer to their parent. Local operations of these types are applied iteratively as long as we find an operation which gives enough progress. Similarly, we can reduce the power consumption by choosing a smaller inverter or buffer for each tree node with the property that no sink of the corresponding subtree has a negative slack. Clearly, each exchange operation has to check that the overall slack is not decreased.

4 Computational Experiments

In this section we report on our computational experience with the proposed methods from the previous sections. Due to space limitations we focus on solution quality only.

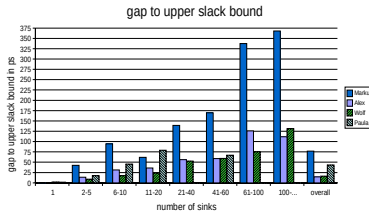


Fig. 4. The average gap in picoseconds of the slack achieved in our heuristic to the upper slack bound.

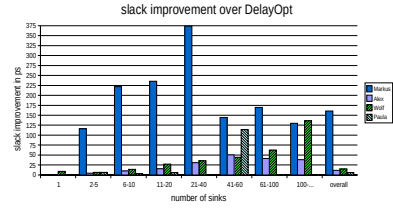


Fig. 5. The average slack improvement in picoseconds achieved in our heuristic in comparison with DelayOpt.

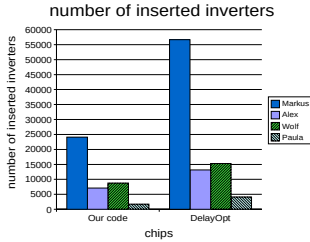


Fig. 6. The number of inserted inverters by our heuristic and by DelayOpt.

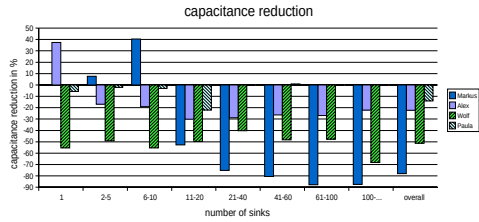


Fig. 7. The average percentage reduction of the input capacitance of inserted inverters achieved by our heuristic in comparison with DelayOpt.

Problem instances and computational set-up. For this study we used four recent ASIC designs from our cooperation partner IBM. All our instances have been extracted from the original design flow. The size of these chips ranges from 1.0 million up to 3.8 million circuits, and from .7 million up to 3.0 million nets. For proprietary reasons we use the code-names Markus, Wolf, Paula, and Alex to refer to the chips. The instances of the four chips have very different characteristics, for example, due to differences in the size and distribution of blockages and to the distribution of the sinks of an instance over the placement area. As a consequence, we evaluate our computational results individually for the four test chips. The range of our test instances is from a single sink up to 206 sinks. Figure 2 shows the distribution of test instances with respect to the number of sinks. The clear majority of instances has only a small number of sinks. A typical instance with relatively many sinks is shown in Figure 1. The experiments are all run on a IBM S85 machine with 16 processors and 96 GB main memory. Our code is implemented in C++ and compiled with the VAC-compiler under the operating system AIX 5.1. We compare our code with a standard tool, called *DelayOpt*, which has been used by IBM for the physical design of these chips.

Wire length. The wire length of the buffered Steiner trees is a good indicator for the enormous differences of the data profile on the four chips. Namely, we find that the tree length on chip Markus is on average about twenty times

longer than on the three other chips. As a lower bound for the necessary wire length we simply use the length of a Steiner minimum tree taking the sinks and the root as terminals. In this Steiner minimum tree computation all blockages have been ignored. Figure 3 shows that our trees are less than twice as long as the lower bound on chip Markus, and even much closer to this bound for the other chips. We also clearly improve the tree lengths in comparison with DelayOpt.

Gap to the upper slack bound. We compare the slack achieved by our approach with the upper slack bounds computed by the method as described in Section 2. Figure 4 shows the gap in picoseconds between these two values, averaged over the different instance classes. It turns out that the average gap to the upper slack bound is relatively small. Not very surprisingly the gap increases with the number of sinks. Considerable gaps occur for instances with more than 60 sinks on chip Markus.

Comparison with DelayOpt. We compare the results for our code with those achieved by DelayOpt. The experiments clearly indicate that our code can consistently improve the slack in comparison with DelayOpt, the largest average improvements have been possible for chip Markus, see Figure 5. For those cases where we can only slightly improve the slack, we achieve, however, big savings in the capacitance of inserted inverters, see Figure 7. We also drop the number of inserted inverters by roughly one half as can be seen in Figure 6. Note that both codes use only inverters but no buffers for insertion.

More results, in particular, a detailed analysis of the impact of several heuristic variants, will appear in the journal version of this paper. Finally, we note that our code solves each instance within a few seconds.

References

1. C. J. Alpert and A. Devgan, *Wire segmenting for improved buffer insertion*, Proceedings of the 34th Design Automation Conference, 1995, pp. 588–593.
2. C. J. Alpert, G. Gandham, M. Hrkic, J. Hu, A. B. Kahng, J. Lillis, B. Liu, S. T. Quay, S. S. Sapatnekar, and A. J. Sullivan, *Buffered Steiner trees for difficult instances*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **21** (2002), 3–13.
3. C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng, and D. Karger, *Prim-Dijkstra tradeoffs for improved performance-driven routing tree design*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **14** (1995), 890–896.
4. C. C. N. Chu and D. F. Wong, *Closed form solutions to simultaneous buffer insertion/sizing and wire sizing*, Proceedings of ISPD, 1997, pp. 192–197.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph drawing: Algorithms for the visualization of graphs*, Prentice Hall, 1999.
6. M. R. Garey and D. S. Johnson, *The rectilinear Steiner tree problem is NP-complete*, SIAM Journal on Applied Mathematics **32** (1977), 826–834.
7. M. Hanan, *On Steiner's problem with rectilinear distance*, SIAM Journal on Applied Mathematics **14** (1966), 255–265.

8. L.-D. Huang, M. Lai, D. F. Wong, and Y. Gao, *Maze routing with buffer insertion under transition time constraints*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **22** (2003), 91–96.
9. I. Klick, *Das Inverterbaum-Problem im VLSI-Design*, Diplomarbeit, Research Institute for Discrete Mathematics, Bonn, 2001.
10. M. Lai and D. F. Wong, *Maze routing with buffer insertion and wiresizing*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **21** (2002), 1205–1209.
11. J. Lillis, C. K. Cheng, and T. Y. Lin, *Optimal wire sizing and buffer insertion for low and a generalized delay model*, IEEE Journal of Solid-State Circuits **31** (1996), 437–447.
12. X. Tang, R. Tian, H. Xiang, and D. F. Wong, *A new algorithm for routing tree construction with buffer insertion and wire sizing under obstacle constraints*, Proceedings of ICCAD-01, 2001, pp. 49–56.
13. L.P.P.P. van Ginneken, *Buffer placement in distributed RC-trees networks for minimal Elmore delay*, Proceedings of the IEEE International Symposium on Circuits and Systems, 1990, pp. 865–868.
14. M. Zachariasen, *A catalog of Hanan grid problems*, Networks **38** (2001), 76–83.
15. H. Zhou, D. F. Wong, I-M. Liu, and A. Aziz, *Simultaneous routing and buffer insertion with restrictions on buffer locations*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **19** (2000), 819–824.

Multisampling: A New Approach to Uniform Sampling and Approximate Counting

Piotr Sankowski

Institute of Informatics
Warsaw University, ul. Banacha 2, 02-097 Warsaw
`sank@mimuw.edu.pl`

Abstract. In this paper we present a new approach to uniform sampling and approximate counting. The presented method is called multisampling and is a generalization of the *importance sampling* technique. It has the same advantage as *importance sampling*, it is unbiased, but in contrary to its prototype it is also an *almost uniform sampler*. The approach seems to be as universal as Markov Chain Monte Carlo approach, but simpler. Here we report very promising test results of using multisampling to the following problems: counting matchings in graphs, counting colorings of graphs, counting independent sets in graphs, counting solutions to knapsack problem, counting elements in graph matroids and computing the partition function of the Ising model.

1 Introduction

The algorithms for approximate counting, such as the Markov Chain Monte Carlo method, find applications in statistical physics. The usefulness of such algorithms for physicists is determined by the practical efficiency. Physicists are usually not interested in theoretical results concerning the precision of the algorithm. The validity of their computations is verified by statistical analysis in the same way as normal experiments are. From the other side it is unclear if the theoretically efficient algorithms – FPRAS'es (*fully polynomial randomized approximation schemes*) – are practically useful. Although such algorithms have polynomial time complexity, the degree of the polynomial is usually too high for practical applications even for small problem instances.

In this paper we generalize the *importance sampling* method [16,3,18]. We show for the first time how the method can be used to construct *almost uniform samplers*. We achieve this by generating a set of samples, instead of one at a time as in *importance sampling* or in Markov Chain approach. The more samples we take the closer their distribution is to uniform distribution. We show also how to use this sampler to construct unbiased estimators.

We describe the method for problems with inheritance property but it can also be applied to self-reducible problems. Inheritance means that every subset of a problem solution is also a correct solution of the problem. In particular the following problems can be defined in such a way that they have this property: matchings in graphs, colorings of graphs, independent sets in graphs, knapsack

problem solutions, matroids and Ising system. Counting solutions to every of these problems is $\#P$ -complete thus exact polynomial time algorithms most probably do not exist. Ising model and some other of these problems have direct physical applications: matchings to model monomer-dimer systems, colorings to the Potts model, independent sets to hard-core gas.

For every of these counting problems an FPRAS in a special case or even in a general case is known. There is an FPRAS for:

- counting the number of all matchings in graphs given by Jerrum and Sinclair [9],
- counting the number of $2\Delta + 1$ colorings in graphs with degree at most Δ presented by Jerrum [11],
- counting the number of independent sets in graphs with maximal degree $\Delta \leq 5$ proved by Dyer and Greenhill [5],
- counting the number of elements in balanced matroids shown by Feder and Mihail [6],
- counting the number of solutions of a knapsack problem found by Morris and Sinclair [17],
- computing the partition function of the ferromagnetic Ising model given by Jerrum and Sinclair [10].

We compare in tests the multisampling with all the above algorithms. We give the same computational time to both methods and measure the errors of estimates they give. In most cases the multisampling performs better than the algorithms based on Markov Chains. Multisampling is also simpler to implement.

2 Definitions

Let Ω denote a set and let $f : \Omega \rightarrow R$ be a function whose values we want to compute. A *randomized approximation scheme* for f is a randomized algorithm that takes $x \in \Omega$ and $\epsilon > 0$ as an input and returns a number Y (the value of a random variable) such that

$$P((1 - \epsilon)f(x) \leq Y \leq (1 + \epsilon)f(x)) \geq \frac{3}{4}.$$

We say that a *randomized approximation scheme* is *fully polynomial* if it works in time polynomially dependent on the size of input data x and ϵ^{-1} . Algorithms that approximate numerical values are called estimators. An estimator is unbiased if it's result is a random variable with expected value equal to the value being approximated.

The total variation distance of two distributions π, ρ over a set Ω is given by

$$\|\pi - \rho\|_{tv} = \sup_{A \subset \Omega} |\pi(A) - \rho(A)|.$$

In this paper we will use this definition but in an extended meaning. For one of the distribution it may happen that $\rho(\Omega) \leq 1$. The quantity $1 - \rho(\Omega)$ is

interpreted as the probability that a sampling algorithm for ρ is allowed to fail and not to generate any sample.

Let $\mathcal{S}$ be a function that for a problem instance x gives the set of all possible solutions to x . An *almost uniform sampler* for $\mathcal{S}$ is a randomized algorithm that takes as an input x and tolerance $\delta > 0$ and returns an element $X \in \mathcal{S}(x)$, such that total variation distance of distribution of X from the uniform distribution is smaller than δ . We say that an *almost uniform sampler* is *fully polynomial* if it works in time polynomially dependent on the size of input data x and $\log \delta^{-1}$.

3 Importance Sampling

Let us suppose we want to compute the size of a finite set Ω . If we are able to generate elements of Ω randomly with some known distribution π , we can write the size of Ω by a simple formula

$$|\Omega| = \sum_{X \in \Omega} 1 = \sum_{X \in \Omega} \pi(X) \frac{1}{\pi(X)}. \quad (1)$$

The distribution π can be arbitrary but must guarantee that every element is generated with non-zero probability. In other words the size of Ω is given by the average of $1/\pi(X)$ over the distribution π . Notice that this equation is correct even if $\pi(\Omega) < 1$. In such a case we allow our estimator to fail. When it cannot generate a sample from Ω it returns zero. The variance of the method is given by the formula

$$\text{Var}(\pi) = \sum_{X \in \Omega} \pi(X) \frac{1}{\pi(X)^2} - |\Omega|^2 = \sum_{X \in \Omega} \frac{1}{\pi(X)} - |\Omega|^2 \leq |\Omega| \max_{X \in \Omega} \frac{1}{\pi(X)}. \quad (2)$$

We see that the variance is small if the distribution π is close to the uniform distribution. For the illustration of the method in the case of counting matchings see [18,3].

4 Multisampling

The limited usefulness of importance sampling follows from the fact that we cannot change the precision of the method. The basic approach that gives such possibility is the idea of *almost uniform samplers*. However the *importance sampling* as presented in previous section does not give an *almost uniform sampler*. In this section we show how such sampler can be constructed. The main idea is to use a set of samples to approximate the uniform distribution instead of using one sample from almost uniform distribution.

Definition 1. A problem with inheritance property is a pair $P = (E, \mathcal{S})$ that fulfills the following conditions

1. E is a finite not empty set.

2. $\mathcal{S}$ is a nonempty family of subsets of E , such that if $B \in \mathcal{S}$ and $A \subset B$ then $A \in \mathcal{S}$. Empty set $\emptyset$ always belongs to $\mathcal{S}$.

We call elements of E solution elements, and the elements of the set $\mathcal{S}$ – solutions. For matchings E is the set of edges of the graph, for independent sets it is the set of vertices. We denote by m the size of the set E .

Let us denote by Ω^k the set of solutions of cardinality k . We can obtain elements of the set Ω^{k+1} by adding elements of E to elements of Ω^k . For $x \in \Omega^k$ we denote by $S(x)$ the set of elements from Ω^{k+1} that can be constructed from x , i.e. $S(x) = \{x \cup \{e\} : x \cup \{e\} \in \Omega^{k+1}, e \in E\}$. We also write $s(x) = |S(x)|$.

Notice that if we know how to generate elements from Ω^k with uniform distribution than we also know how to generate elements from Ω^{k+1} with almost uniform distribution. We generate many elements from Ω^k , next we choose from them an element x with probability proportional to $s(x)$ and generate uniformly at random an element from $S(x)$.

In multisampling we generate N_{j+1} samples from the set Ω^{j+1} by using N_j samples from Ω^j which were generated in the previous step of the algorithm. The samples are generated in arrays X^j with elements $X_i^j \in \Omega^j$, for $1 \leq i \leq N_k$ (see Algorithm 4.1).

Algorithm 4.1 Multisampling Algorithm

```

for  $i := 1$  to  $N_0$  do
   $X_i^0 :=$  empty solution
end for
for  $j := 0$  to  $k - 1$  do
  for  $l := 1$  to  $N_{j+1}$  do
    choose from  $X^j$  a sample  $x$  with probability proportional to  $s(x)$ 
    generate uniformly at random an element from  $S(x)$  and put it into  $X_l^{j+1}$ 
  end for
end for

```

4.1 Almost Uniform Sampler

Let us denote by $\Pr(X_i^k = x)$ the probability of obtaining the sample x on place i in the array X^k . It is easy to see the following.

Remark 1. Let $k \geq 0$, $\forall_{0 \leq j < k} N_j = N$ and $x \in \Omega^k$ then,

$$\lim_{N \rightarrow \infty} \Pr(X_i^k = x) = \frac{1}{|\Omega^k|}.$$

The Remark says that the method is an *almost uniform sampler*. Although the algorithm is pretty simple we have been unable to show satisfactory bounds on the convergence of the method. However the test and some partial theoretical results show that the total variation distance of the distribution of $\Pr(X_i^k = \cdot)$ from the uniform distribution is proportional to $\frac{1}{N} - |\Pr(X_i^k = \cdot) - \frac{1}{|\Omega^k|}|_{tv} \sim \frac{1}{N}$.

4.2 Multisampling and Counting

As we stated in the introduction the multisampling can be used to construct unbiased estimators. The following procedure can be used to approximate the size of the set $|\Omega^k|$ (see Algorithm 4.2).

Algorithm 4.2 Estimator Based on Multisampling

```

run the Algorithm 4.1
for  $0 \leq j < k$  do
   $q_j = \frac{1}{N_j} \sum_{1 \leq i \leq N_j} \frac{s(X_i^j)}{j+1},$ 
end for
return  $z_k = q_0 q_1 \dots q_{k-1}$ 
  
```

The next theorem states that this estimator is unbiased.

Theorem 1. *Let $k \geq 0$ and $\forall_{0 \leq j < k} N_j > 0$ then $E(z_k) = |\Omega^k|$.*

Proof. The proof of the theorem is simple when we notice that this method is an importance sampling method. For every j the sample X_1^{j+1} was obtained from some sample X_p^j from the previous generation. The probability of obtaining generation of samples X^j is independent from the ordering of samples so we may swap the sample X_p^j with the sample X_1^j and multiply the probability by N_j - the number of possibilities for p . Let us now fix other samples in X^j . We have

$$\begin{aligned}
 \Pr(X_1^{j+1} = x) &= \sum_{y \in \Omega^j} \sum_{e \in E} \Pr(X_1^j = y) \frac{N_j[x = y \cup e]}{s(y)} \frac{s(y)}{\sum_{r=1}^{N_j} s(X_r^j)} = \\
 &= \sum_{y \in \Omega^j} \sum_{e \in E} \Pr(X_1^j = y) \frac{N_j[x = y \cup e]}{\sum_{r=1}^{N_j} s(X_r^j)} = \sum_{y \in \Omega^j} \sum_{e \in E} \Pr(X_1^j = y) \frac{N_j[y = x - e]}{\sum_{r=1}^{N_j} s(X_r^j)} = \\
 &= \sum_{e \in E} [x - e \in \Omega^j] \Pr(X_1^j = x - e) \frac{N_j}{\sum_{r=1}^{N_j} s(X_r^j)} = \\
 &= \sum_{e \in x} \Pr(X_1^j = x - e) \frac{N_j}{\sum_{r=1}^{N_j} s(X_r^j)} = \frac{1}{j+1} \sum_{e \in x} \Pr(X_1^j = x - e) \frac{N_j(j+1)}{\sum_{r=1}^{N_j} s(X_r^j)} \\
 &= \frac{1}{j+1} \sum_{e \in x} \Pr(X_1^j = x - e) \frac{1}{q^j}.
 \end{aligned}$$

If we apply the above equation recursively to $\Pr(X_1^k = x)$, we get

$$\Pr(X_1^k = x) = \frac{1}{k} \sum_{e_k \in x} \frac{1}{k-1} \sum_{e_{k-1} \in x - e_k} \frac{1}{k-2} \sum_{e_{k-2} \in x - e_k - e_{k-1}} \dots$$

$$\frac{1}{2} \sum_{e_2 \in x - e_k - e_{k-1} - \dots - e_3} \frac{1}{1} \sum_{e_1 \in x - e_k - e_{k-1} - \dots - e_3 - e_2} \frac{1}{z^k}.$$

From above equation you may notice that z_k is an inverted probability of generating the sample x in generation X_k in the case when choices of e are fixed and also all other samples are fixed. We now apply (1) and get that expected value of z_k is equal to the size of the set Ω_k and the theorem follows. $\square$

The following bound on the variance of the method can be obtained from (2).

Theorem 2. *The variance of Estimator 4.2 fulfills $\text{Var}(z_k) \leq \binom{|E|}{k} E(z_k)$.*

This bound is not satisfactory as it does not give a polynomial time algorithms for counting.

4.3 Multisampling and Nonuniform Distributions

Multisampling can be also used to sample from nonuniform distribution and for computing the partition functions. Let us assume that we are given a probability distribution ϕ over all sets Ω^i . To adopt multisampling for sampling from this distribution we have to change the definition of $s(x)$ to

$$s(x) = \sum_{y \in S(x)} \frac{\phi(y)}{\phi(x)},$$

and to change multisampling is the following way (see Algorithm 4.3).

Algorithm 4.3 Multisampling Algorithm for Nonuniform Distributions

```

for  $i := 1$  to  $N_0$  do
     $X_i^0 :=$  empty solution
end for
for  $j := 0$  to  $k - 1$  do
    for  $l := 1$  to  $N_{j+1}$  do
        choose from  $X^j$  a sample  $x$  with probability proportional to  $s(x)$ 
        generate an element  $y \in S(x)$  with the probability proportional to  $\frac{s(y)}{s(x)}$  and put
        it into  $X_l^{j+1}$ 
    end for
end for
    
```

The partition function over the set Ω^k is given by

$$Z_k = \sum_{x \in \Omega^k} \phi(x).$$

Algorithm 4.2 can be applied without any modification to the case of nonuniform distributions. The next theorem can be proved in the same way as Theorem 1.

Theorem 3. *Let $k \geq 0$ and $\forall_{0 \leq j < k} N_j > 0$ then $E(z_k) = Z_k$.*

5 Tests

We have compared the algorithms by giving them the same time and measuring the average relative error of the results they give. The error was computed over 30 independent runs and the algorithms were given 5 second of time on a personal computer (Athlon 1.2GHz, 512MB RAM). For all investigated counting problems multisampling gave results with errors smaller than Markov Chain Monte Carlo method. For graph problems as matchings, independent sets, forests, colorings and Ising model, we have made tests on: two and three dimensional grids, random sparse graphs and random dense graphs. The results we obtained were similar for all types of graphs. Here we only report results for two dimensional rectangular grids. There are two reasons: firstly regular graphs are a worst case for randomized algorithms, and secondly grids are the most interesting from the physical point of view.

In multisampling for simplicity we have used the same number N of samples in all generations. An especially chosen sequence N_j could speed up the algorithm. The number N was adjusted for each test separately, in such a way that the algorithm fitted exactly in the time limit. The selection of the samples with probability proportional to $s(x)$ was implemented using binary trees. If we denote by f the time needed to compute $s(x)$ and to generate a random sample from $S(x)$ then each step of the algorithm works in time $O(N \ln N + Nf)$.

Due to space limit here we give only detailed results of the tests for three problems: counting matchings in graphs, counting colorings and Ising model. The results for other three problems: counting knapsack solutions, counting forests and counting independent sets will be included in the full version of the paper.

5.1 Matchings in Graphs

To use Markov Chain for counting matchings in a given graph G we use the partition function approach. The partition function is defined by

$$Z(\lambda) = \sum_{M \text{ matching}} \lambda^{|M|} = \sum_{k=0}^{\frac{n}{2}} m_k \lambda^k,$$

where m_k is the number of matchings of cardinality k in graph G . For $\lambda = 1$ the value of Z is equal to the number of all matchings in G . To guarantee the accuracy of the procedure we have to express $Z(1)$ as the product

$$Z(1) = \frac{Z(\lambda_r)}{Z(\lambda_{r-1})} \times \frac{Z(\lambda_{r-1})}{Z(\lambda_{r-2})} \times \cdots \times \frac{Z(\lambda_2)}{Z(\lambda_1)} \times \frac{Z(\lambda_1)}{Z(\lambda_0)} \times Z(\lambda_0).$$

where $0 = \lambda_0 < \lambda_1 < \lambda_2 < \cdots < \lambda_{r-1} < \lambda_r = 1$ is a suitably chosen sequence of values. We have $Z(\lambda_0) = Z(0) = 1$. The reciprocal of the ratio $Z(\lambda_i)/Z(\lambda_{i-1})$ is equal to the expectation of $f_i(M) = \left(\frac{\lambda_{i-1}}{\lambda_i}\right)^{|M|}$ over the distribution π_{λ_i} given by

$$\pi_{\lambda}(M) = \frac{\lambda^{|M|}}{Z(\lambda)}.$$

We have

$$E(f_i) = \sum_M \left(\frac{\lambda_{i-1}}{\lambda_i} \right)^{|M|} \frac{\lambda^{|M|}}{Z(\lambda)} = \frac{1}{Z(\lambda_i)} \sum_M \lambda_{i-1}^{|M|} = \frac{Z(\lambda_{i-1})}{Z(\lambda_i)}.$$

In order to estimate $\frac{Z(\lambda_{i-1})}{Z(\lambda_i)}$ we sample matchings with distribution π_λ using the chain given by Jerrum and Sinclair in [12] (see Algorithm 5.1).

Algorithm 5.1 Markov Chain for Matchings

with probability $\frac{1}{2}$ let $M' = M$; otherwise,
select an edge $e = \{u, v\} \in E$ uniformly at random
and set

$$M' = \begin{cases} M - e & \text{if } e \in M; \\ M + e & \text{if both } u \text{ and } v \text{ are not matched} \\ & \text{in } M; \\ M + e - e' & \text{if one of } u \text{ and } v \text{ is matched in} \\ & M \text{ and } e' \text{ is the matching edge;} \\ M & \text{otherwise;} \end{cases}$$

go to M' with probability $\min(1, \pi_\lambda(M')/\pi_\lambda(M))$.

We compare multisampling with three different algorithms using this partition function framework:

- (a) we use $\lambda_1 = |E|^{-1}$ and $\lambda_{i+1} = \left(1 + \frac{1}{n}\right) \lambda_i$, and independent runs of the chain for each sample;
- (b) we use $\lambda_1 = |E|^{-1}$ and $\lambda_{i+1} = \left(1 + \frac{1}{n}\right) \lambda_i$, and we average over trajectory of the chain;
- (c) we use $\lambda_1 = |E|^{-1}$ and $\lambda_{i+1} = \left(1 + \frac{1}{n}\right)^{\sqrt{n}} \lambda_i$, and we average over trajectory of the chain.

The choice of λ_i in (a) and (b) guarantees that the ratios $\frac{Z(\lambda_{i-1})}{Z(\lambda_i)}$ are small (for details see [12]), whereas the sequence in (c) is chosen empirically.

One step of the chain was implemented in time $O(1)$ and f was of the order $O(m)$. Results of the tests are shown in Fig. 5.1.

5.2 Colorings

To use multisampling to sample colorings we have to define them in such a way that inheritance property holds. Thus we define coloring as a set of ordered pairs (v, c) where $v \in V$ is a vertex and $1 \leq c \leq k$ is its color.

In the case of colorings we cannot use partition function approach because all colorings are of the same cardinality. We use similar approach. Let $E =$

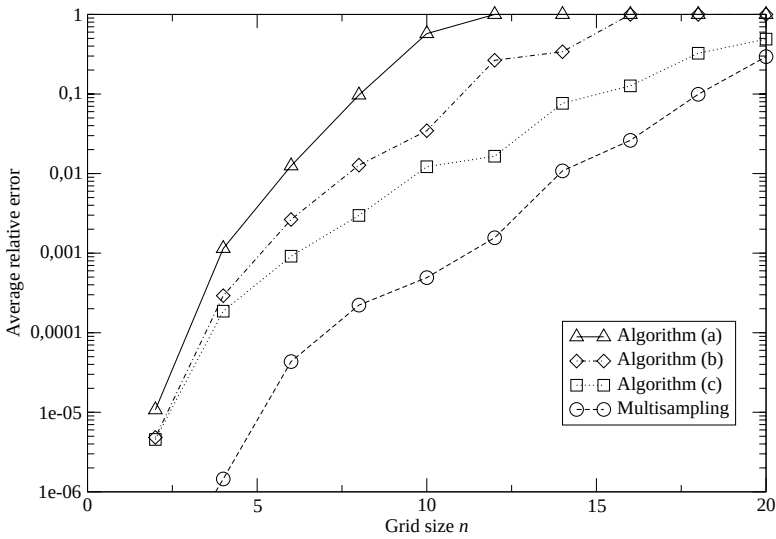


Fig. 1. Comparison of the methods for counting matchings for two dimensional rectangular grids of size $n \times n$.

$\{e_1, \dots, e_m\}$ be the set of edges in the graph. Let us denote by $Z(A)$ the number of colorings in a graph with edge set A . We express $Z(E)$ as the product

$$Z(E) = \frac{Z(\{e_1, \dots, e_m\})}{Z(\{e_1, \dots, e_{m-1}\})} \times \frac{Z(\{e_1, \dots, e_{m-1}\})}{Z(\{e_1, \dots, e_{m-2}\})} \times \dots \\ \times \frac{Z(\{e_1, e_2\})}{Z(\{e_1\})} \times \frac{Z(\{e_1\})}{Z(\emptyset)} \times Z(\emptyset).$$

You may notice that $Z(\emptyset) = k^n$. We can now approximate each of the ratios $\frac{Z(\{e_1, \dots, e_i\})}{Z(\{e_1, \dots, e_{i-1}\})}$ by sampling colorings of the graph with edge set $\{e_1, \dots, e_{i-1}\}$ and checking how many of them are correct colorings of the graph with edge set $\{e_1, \dots, e_i\}$.

In this case the choice of the ratios is fixed, so we use only two algorithms based on Markov Chains:

- (d) we use independent runs of the chain for each sample;
- (c) we average over trajectory of the chain.

To sample k -colorings in graphs with maximum degree Δ , where $k \geq 2\Delta + 1$, we use following Markov Chain (see Algorithm 5.2).

One step of the chain was implemented in time $O(k)$, where k is the number of used colors and f was of the order $O(m + k)$. For results of the tests see Fig. 5.2.

Algorithm 5.2 Markov Chain for Colorings

with probability $\frac{1}{2}$ let $C' = C$; otherwise,
 select a vertex $v \in V$ uniformly at random,
 set C' to the coloring obtained from C by recoloring v to a free color (a free color
 always exists).

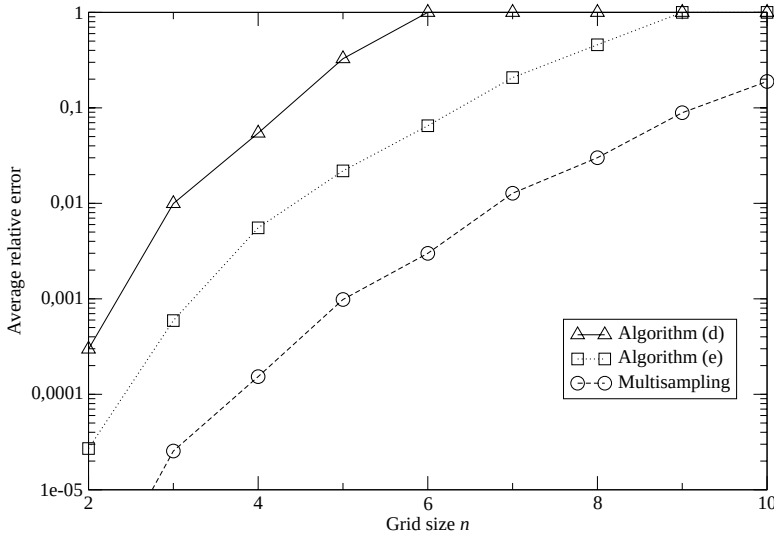


Fig. 2. Comparison of the methods for counting 9 colorings for two dimensional rectangular grids of size $n \times n$.

5.3 Ising Model

We define the state of the Ising model for a graph $G = (E, V)$ as the vector $x_i \in \{-1, 1\}$, for $i \in V$. The energy of a state x is given by

$$\mathcal{E}_E(x) = -J \sum_{(i,j) \in E} x_i x_j,$$

where J is a coupling constant. We are interested in computing the partition function of the system given by

$$Z(E) = \sum_{x \in \{-1, 1\}^V} e^{-\mathcal{E}_E(x)}.$$

For computing the partition function of the Ising model we use the same approach as for colorings. The only difference is that now to approximate the ratio $\frac{Z(\{e_1, \dots, e_i\})}{Z(\{e_1, \dots, e_{i-1}\})}$ we average the random variable

$$f_i(x) = \frac{\exp(-\mathcal{E}_{\{e_1, \dots, e_i\}}(x))}{\exp(-\mathcal{E}_{\{e_1, \dots, e_{i-1}\}}(x))} = \exp(-\mathcal{E}_{\{e_i\}}(x))$$

over the configurations of the Ising model given by the edge set $\{e_1, \dots, e_{i-1}\}$.

To use multisampling to the Ising model we proceed in the same way as for colorings. We consider the set of ordered pairs (v, s) , where $v \in V$ is a vertex and $s \in \{-1, 1\}$ is a spin in this vertex. Notice that the value $s(x)$ is simple to compute, as the ratios $\frac{\phi(y)}{\phi(x)}$ are given by the change of the energy when a new spin is added.

One step of the chain was implemented in time $O(n)$ and f was of the order $O(m + k)$. Results of the tests are shown in Fig. 5.3.

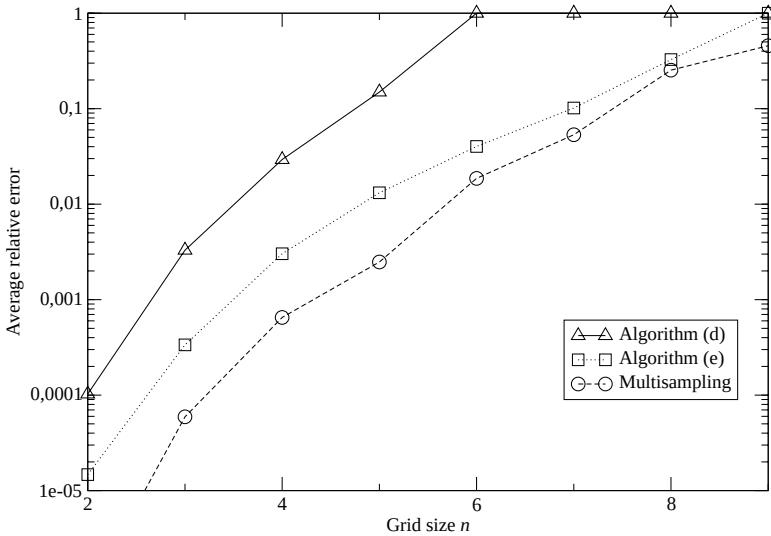


Fig. 3. Comparison of the methods for computing the partition function of the Ising model for two dimensional rectangular grids of size $n \times n$ with the coupling constant $J = 0.3$.

6 Summary

The tests show that the multisampling gives better results than Markov Chain Monte Carlo approach, and it is also simpler. Applying Markov Chains for counting we have to express the approximated value as the product of some smaller ones, when the multisampling can be applied directly – Algorithm 4.2. Thus the multisampling gives an interesting alternative to the Markov Chain that is worth investigating. Unfortunately although the method is pretty simple it is difficult to analyze theoretically – works are in progress.

References

1. Alexander Barvinok, Polynomial time algorithms to approximate permanents and mixed discriminants within a simply exponential factor, *Random Structures and Algorithms* **14** (1999), 29–61.

2. Alexander Barvinok, New Permanent Estimators Via Non-Commutative Determinants, preprint.
3. Isabel Beichl and Francis Sullivan, Approximating the Permanent via Importance Sampling with Application to Dimer Covering Problem, *Journal of computational Physics* **149**, 1, February 1999.
4. Steve Chien, Lars Rasmussen and Alistair Sinclair, Clifford Algebras and Approximating the Permanent, *Proceedings of the 34th ACM Symposium on Theory of Computing*, 2001, 712–721.
5. Martin Dyer and Catherine Greenhill, On Markov Chains for independent sets. preprint 1997.
6. T. Feder and M. Mihail. Balanced matroids, Proceedings of the 24th Annual ACM Symposium on the theory of Computing (STOC), (1992), 26–38.
7. A. Frieze and M. Jerrum, An analysis of a Monte Carlo algorithm for estimating the permanent, *Combinatorica*, **15** (1995), 67–83.
8. C.D. Godsil and I. Gutman, On the matching polynomial of a graph, *Algebraic Methods in Graph Theory, Vol. I, II, (Szeged, 1978)*, North-Holland, Amsterdam – New York, 1981, 241–249.
9. Mark Jerrum and Alistair Sinclair, Approximating the permanent, *SIAM Journal on Computing* **18** (1989), 1149–1178.
10. Mark Jerrum and Alistair Sinclair, Polynomial-time Approximation Algorithms for the Ising Model, *SIAM Journal on Computing* **22** (1993), 1087–1116.
11. Mark Jerrum, A very simple algorithm for estimating the number of k -colorings of a low-degree graph, *Random Structures and Algorithms* **7** (1995), 157–165.
12. Mark Jerrum and Alistair Sinclair, The Markov Chain Monte Carlo method: an approach to approximate counting and integration, in *Approximation Algorithms for NP-hard Problems* (Dorit Hochbaum, ed.), PWS 1996.
13. Mark Jerrum, Alistair Sinclair and Eric Vigoda, A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries, STOC, 2000.
14. N. Karmarkar, R. Karp, R. Lipton, L. Lovász and M. Luby, A Monte Carlo algorithm for estimating the permanent, *SIAM Journal on Computing* **22** (1993), 284–293.
15. E. Kaltofen and G. Villard, On the complexity of computing determinants, *Proc. Fifth Asian Symposium on Computer Mathematics (ASCM 2001)*, volume **9** of Lecture Notes Series on Computing, pages 13–27, Singapore, 2001.
16. Lars Eilstrup Rasmussen, Approximating the Permanent: a Simple Approach, *Random Structures Algorithms* **5** (1994), 349–361.
17. Ben Morris and Alistair Sinclair, Random Walks on Truncated Cubes and Sampling 0–1 Knapsack Solutions, Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS), (1999), 203–240.
18. Piotr Sankowski, Alternative Algorithm for Counting All Matchings in Graphs, Proceedings of the 20th International Symposium on Theoretical Aspects of Computer Science (STACS), (2003), LNCS 2607, 427.

Multicommodity Flow Approximation Used for Exact Graph Partitioning^{*}

Meinolf Sellmann¹, Norbert Sensen², and Larissa Timajev²

¹ Cornell University, Department of Computer Science
4130 Upson Hall, Ithaca, NY 14853, U.S.A.
sello@cs.cornell.edu

² University of Paderborn
Faculty of Computer Science, Electrical Engineering, and Mathematics
Fürstenallee 11, 33102 Paderborn, Germany
{sensen,timajev}@upb.de

Abstract. We present a fully polynomial-time approximation scheme for a multi-commodity flow problem that yields lower bounds of the graph bisection problem. We compare the approximation algorithm with Lagrangian relaxation based cost-decomposition approaches and linear programming software when embedded in an exact branch&bound approach for graph bisection. It is shown that the approximation algorithm is clearly superior in this context. Furthermore, we present a new practical addition to the approximation algorithm which improves its performance distinctly. Finally, we prove the performance of the graph bisection algorithm using multicommodity flow approximation by computing formerly unknown bisection widths of some DeBruijn- and Shuffle-Exchange-Graphs.

1 Introduction

We consider the graph bisection problem that consists in partitioning the nodes of an arbitrary undirected graph into two sets of equal size such that the (possibly weighted) sum of edges that cross the cut is minimized. The problem is known to be NP-hard. Based on our previous work presented in [31], we develop an algorithm that, given a graph, determines its exact bisection width which is defined as the value of the smallest balanced cut. Because of its inherent hardness, we cannot hope for an efficient algorithm to tackle the problem (at least in terms of worst case running times and under the common assumption that $NP \neq P$), but we aim at increasing the size of instances that can still be solved in a reasonable amount of time.

As it is the case for any combinatorial optimization problem, the task of computing the bisection width of a graph is twofold. First, an optimal solution to the problem must be computed, and second, its optimality must be proved. Regarding the first task, efficient heuristics have been developed in the literature for computing high quality and, for small

^{*} This work was partly supported by the German Science Foundation (DFG) project SFB-376, by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), and by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076).

graphs, often even optimal solutions very quickly [14,19,24,26]. To prove the optimality of a given solution, we use a total enumeration branch&bound-approach. The key issue for the development of a good tree search algorithm is to compute tight lower bounds on the bisection width.

In the last years, a number of exact approaches for solving the graph partitioning problem have been presented. The most recent approach is presented in [18] by S.E. Karisch, F. Rendl, and J. Clausen who use semidefinite programming relaxations for computing bounds on the objective. In [6], Ferreira et al. present a branch-and-cut algorithm for the problem. The same approach is followed by L. Brunetta, M. Conforti, and G. Rinaldi in [2]. In [15], E. Johnson, A. Mehrotra, and G. Nemhauser elaborate on a column generation approach for the graph partitioning problem.

In [31], we presented a new lower bound for the bisection width of a graph. It can be obtained by solving a multicommodity flow problem with variable sender volumes. The problem may be viewed as a generalized maximum multicommodity flow problem: we need to compute a maximum multicommodity flow on an undirected network where the commodities have exactly one source, but may have many sinks. Building up on existing techniques for the solution of multicommodity flow problems, we develop a fully polynomial time approximation scheme (FPTAS). The routine is embedded in a branch&bound-framework for exact graph bisection and experimentally compared with a Lagrangian relaxation based cost-decomposition approach and a barrier LP-solver on various test instances. On top of that, we compare the multicommodity bound and the different algorithms for its computation/approximation with the semidefinite bounding routine developed in [18]. Finally, we prove the performance of our exact graph bisection algorithm by computing the previously unknown bisection width of some DeBruijn- and Shuffle-Exchange-Graphs.

The paper is structured as follows: In Section 2, we review the lower bounds on the bisection width that have been proposed in [31], especially the so called *VarMC*-bound. In Section 3, we develop an algorithm for the approximation of that *VarMC*-bound. A complete branch&bound-approach is sketched in Section 4. Finally, in Section 5, we present our empirical evaluation.

2 Bounds on Graph Bisection

Our work bases on the lower bounds on the graph partitioning problem introduced in [31]. In the following, we give a small survey on the main ideas of these bounds: A well known lower bound (see [21]) on the bisection width can be achieved by embedding a clique with the same number of nodes n into the given graph G . When embedding the clique, an edge between two nodes translates into a path in G . The number of paths that use the same edge e is called the *congestion* of e . The edge with the maximum congestion then determines the congestion of the embedding. If the clique can be embedded with congestion C , we know that G has a bisection width of at least $\frac{n^2}{4C}$. Note that the clique embedding can also be viewed as an integer multicommodity flow problem where every vertex sends a commodity of size one to every other vertex.

First, we note that actually we do not need to enforce the integrality constraints on the flows and thereby can strengthen the bound computed¹. We can further improve on this bound by taking two steps of generalization: First, it can be observed that every single-source multicommodity flow instance (with arbitrary demands and destinations) can be used to compute a lower bound on the bisection width. The critical point is the *CutFlow*, i.e. the amount of flow which can be ensured to cross every possible bisection of the graph. It is easy to show that $\frac{\text{CutFlow}}{C}$ is always a valid lower bound on the bisection problem.

Second, we do not have to select an appropriate multicommodity flow instance by ourselves: Typically, linear programming techniques are used to solve multicommodity flow problems. In [31], we introduced the idea to leave the selection of an appropriate multicommodity flow instance to the linear program by adding some variables and constraints. Two different possibilities with a different degree of freedom for the selection of the multicommodity flow instance have been introduced: the *VarMC* and the *MVarMC* formulations. In the *MVarMC* formulation, every node has the freedom to send a commodity of arbitrary size to each other node. Whereas in the *VarMC* formulation, every node has to send a commodity of arbitrary size to every other node, whereby each destination gets the same share. Experiments have shown that, for the graph bisection problem, the *VarMC* formulation gives equally good bounds when compared with the *MVarMC* formulation. Therefore, in the paper at hand, we compare different solution techniques for the computation of the *VarMC*-bound.

3 Approximation of the VarMC-Bound

3.1 The LP-Formulation

Even though the continuous multicommodity flow problem (MCF) is solvable in polynomial time, for more than a decade now researchers have tried to develop approximation schemes for the problem. The reason for this at first surprising fact is that large multicommodity flow instances have to be solved in many application areas and in combination with a whole variety of discrete optimization problems such as network design or graph bisection. And standard simplex or interior point LP-solvers, even specialized software based on primal basis partitioning [4,5] or Lagrangian relaxation based resource- or cost-decompositions [3,9] are simply not fast enough to tackle real-size instances of these problems very efficiently. Therefore, there is a big interest in the development of algorithms that provide near optimal solutions more quickly.

In this section, we develop an ϵ -approximation scheme for the *VarMC*-bound. The space restrictions prevent us from explaining the *VarMC*-bound in more detail. We refer the interested reader to [31]. However, in the following we state a general linear program that generalizes the *VarMC*-bound in the sense that for a given graph the *VarMC*-bound is a special instance of that LP with specific edge-capacities, cost coefficients, and demand values.

Let $G = (V, E)$ denote an undirected network with $|V| = n$, $|E| = m$, with associated node-arc incidence matrix $N \in \{-1, 0, 1\}^{n \times 2m}$, and capacities $u \in \mathbb{R}_{\geq 0}^m$.

¹ For flows, the congestion of an edge is the amount of flow that is routed through it.

Further, let $K \in \mathbb{N}$ denote the number of commodities with source nodes $s^k \in V$ and demands $d^k \in \mathbb{R}^n$, $1 \leq k \leq K$, whereby $d_i^k \geq 0$ for all $i \neq s^k$, and $\sum_i d_i^k = 0$. Finally, denote with p^k the cost coefficient of commodity k . Then, the problem is to

$$\begin{aligned} \text{Maximize} \quad & \sum_k p^k \lambda^k \\ \text{subject to} \quad & Nx^k = \lambda^k d^k \quad \forall 1 \leq k \leq K \quad (1) \\ & \sum_k x_{ij}^k + x_{ji}^k \leq u_{ij} \quad \forall \{i, j\} \in E \quad (2) \\ & \lambda, x \geq 0. \end{aligned}$$

This way to state the problem allows us directly to compute a lower bound on the bisection width of a given graph (compare with Section 2): the objective maximizes the CutFlow, whereby the variables λ^k determine the sender volume of commodity k , and the x^k give the corresponding flow in the network. The constraints (1) ensure that x^k is a feasible flow of commodity k with volume λ^k , and restrictions (2) enforce that the capacity of the undirected edges $\{i, j\}$ is not violated (i.e., the congestion is at most 1).

Note, that to solve the problem it is sufficient to look at the special case with $p^k = 1$ for all $1 \leq k \leq K$, because for $p^k \leq 0$ we set $\lambda^k = 0$ and $x^k = 0$, and otherwise we can scale the demand d^k by $1/p^k$.

3.2 The FPTAS

The first FPTAS's for maximum multicommodity flow were based on Lagrangian relaxations and linear programming. They have been improved and adapted to different models in a series of papers [12,13,17,20,22,25,27,32]. While all this research was built on the idea of rerouting existing flows, the idea of augmenting flow has led to a couple of new algorithms with improved running times [7,8,10,16,34]. Several publications also report about the usability of approximation algorithms for multicommodity flow problems in practice [1,11,13,28,29].

We try to keep the paper self-contained. Note, however, that the following description is analogue to the “maximum multicommodity flow”-section in [8], which itself builds directly on the analysis given in [10]. Our modest contribution here consists in the generalization of the existing algorithms for the maximum multicommodity flow problem to an FPTAS that can handle commodities with more than one sink. This, of course, is essential for the computation of a lower bound for the bisection width of a graph (see Section 2). At the same time, because we focus on graph bisection here, we enable the FPTAS to handle undirected edges. However, the changes that are necessary for that purpose are not of fundamental nature, so that the theory we present can also be used for the development of an approximation scheme for generalized maximum multicommodity flow problems on directed graphs with multi-sink commodities.

For all $1 \leq k \leq K$, denote with $\pi^k(i)$ the set of all paths from the source s^k to the sink $i \in V \setminus \{s^k\}$. Further, set $\pi^k = \cup_{i \neq s^k} \pi^k(i)$. Using variables y_P^k representing the flow of commodity k along some path $P \in \pi^k$, we achieve a *path-based* formulation of the problem:

$$\begin{aligned}
& \text{Maximize} && \sum_k \lambda^k \\
& \text{subject to} && \sum_{P \in \pi^k(i)} y_P^k = \lambda^k d_i^k && \forall 1 \leq k \leq K, i \in V \setminus \{s^k\} \\
& && \sum_{P \in \pi^k: \{i,j\} \in P} y_P^k \leq u_{ij} && \forall \{i,j\} \in E \\
& && \lambda, y \geq 0
\end{aligned}$$

The dual of the above can be written as

$$\begin{aligned}
& \text{Minimize} && \sum_{\{i,j\} \in E} u_{ij} l_{ij} \\
& \text{subject to} && \sum_{\{i,j\} \in P} l_{ij} \geq z_h^k && \forall 1 \leq k \leq K, h \in V \setminus \{s^k\}, P \in \pi^k(h) \\
& && \sum_{i \neq s^k} d_i^k z_i^k \geq 1 && \forall 1 \leq k \leq K \\
& && l \geq 0.
\end{aligned}$$

That is, we have to assign a length $l_{ij} \geq 0$ to each edge $\{i,j\} \in E$, such that $\sum_{i \neq s^k} d_i^k \text{dist}_i^k(l) \geq 1$ for all $1 \leq k \leq K$, and $\sum_{\{i,j\} \in E} u_{ij} l_{ij}$ is minimized, whereby $\text{dist}_i^k(l)$ denotes the shortest path distance from s^k to the sink $i \in V$ in G under length function l .

Consider the approximation scheme sketched in Figure 1. We start with length function $l \equiv \delta$ for an appropriately defined $\delta = \delta(\epsilon, G, d)$, and the primal solution $x \equiv 0$. The length function l is defined on the undirected edge set E , whereas we maintain flow values x_{ij}^k and x_{ji}^k for each edge $\{i,j\} \in E$ and all $1 \leq k \leq K$. While there is still a tree T along which we can route the demand of a commodity k with costs less than 1, the algorithm selects such a tree and augments flow along this tree. More precisely, the algorithm selects a tree with approximately minimal costs up to an approximation factor of $1 + \epsilon$. This property

is achieved by maintaining a lower bound $\hat{\alpha}$ on the current minimal routing costs of any commodity.

```

1:  $x \equiv 0, l \equiv \delta$ 
2:  $\hat{\alpha} \leftarrow \min_k \sum_i d_i^k \delta, \text{oldSink} \leftarrow -1$ 
3: repeat
4:   for all  $1 \leq k \leq K$  do
5:     if  $\text{oldSink} \neq s^k$  then
6:        $T \leftarrow \text{Shortest\_Path\_Tree}(s^k, l)$ 
7:        $\text{oldSink} \leftarrow s^k$ 
8:       while  $\sum_i d_i^k \text{dist}_i^k(l) < \min\{1, (1+\epsilon)\hat{\alpha}\}$  do
9:         for all  $(i,j) \in T$  do
10:           $c_{ij}^k \leftarrow \sum_{h \in T_j} d_h^k$ 
11:           $\phi \leftarrow \min_{(i,j) \in T} (2x_{ji}^k + u_{ij})/c_{ij}^k, \Delta \equiv 0$ 
12:          for all  $(i,j) \in T$  do
13:             $\Delta_{ji} \leftarrow -\min\{x_{ji}^k, \phi c_{ij}^k\}$ 
14:             $\Delta_{ij} \leftarrow \phi c_{ij}^k + \Delta_{ji}$ 
15:            if  $\Delta_{ij} + \Delta_{ji} > 0$  then
16:               $l_{ij} \leftarrow l_{ij}(1 + \epsilon(\Delta_{ij} + \Delta_{ji})/u_{ij})$ 
17:           $x \leftarrow x + \Delta$ 
18:           $T \leftarrow \text{Shortest\_Path\_Tree}(s^k, l)$ 
19:           $\hat{\alpha} \leftarrow \hat{\alpha}(1 + \epsilon)$ 
20: until  $\hat{\alpha} \geq 1$ 

```

Fig. 1. The FPTAS.

The amount of flow sent along tree T is determined in the following way: Denote with T_j all nodes in the subtree routed at node $j \in V$ (including j). For each edge $\{i, j\} \in T$ we compute the congestion c_{ij}^k when routing the demand of commodity k along that tree, i.e., we set $c_{ij}^k = \sum_{h \in T_j} d_h^k$. Basically, we achieve a feasible routing by scaling the flow by $\min_{\{i,j\} \in T} u_{ij}/c_{ij}^k$. However, because we are working on an undirected network here, we would like to consider only flows with $\min\{x_{ij}^k, x_{ji}^k\} = 0$ for all $1 \leq k \leq K$ and $\{i, j\} \in E$. When we also incorporate and change the current flow x_{ji}^k of commodity k in the opposite direction, we achieve an even bigger scaling factor of $\min_{(i,j) \in T} (2x_{ji}^k + u_{ij})/c_{ij}^k$. Formally, we can prove Lemma 1 regarding the change $\Delta_{ij} + \Delta_{ji}$ of the current flow on edge $\{i, j\} \in E$ of commodity k .

In case of a positive flow change on an edge $\{i, j\} \in E$ ($\Delta_{ij} + \Delta_{ji} > 0$), we update the dual variables by setting $l_{ij} \leftarrow (1 + \epsilon(\Delta_{ij} + \Delta_{ji})/u_{ij})l_{ij}$, i.e., we increase the lengths of an edge exponentially with respect to the congestion of that edge. Finally, the primal solution is updated by $x \leftarrow x + \Delta$. This setting may yield an infeasible solution, since it may violate some capacity constraint. However, the mass balance constraints are still valid. This allows us, at the end of the algorithm, to scale the final flows x^k so that they build a feasible solution to the problem.

With the FPTAS in Figure 1, we are able to prove

Theorem 1. *Let $T_{SP} = m + n \log n$. An ϵ -approximate VarMC-bound on the bisection width of a graph can be computed in time $O^*(mT_{SP}/\epsilon^2)^2$.*

Following the analysis in [8], we prove the previous theorem with the help of a sequence of Lemmas³. We set $\rho = \min_{k,i} \{d_i^k \mid d_i^k > 0\}$, and $\sigma = \log_{1+\epsilon}((1+\epsilon)/(\delta\rho))$. Then,

Lemma 1. *In every iteration, in which the current flow is changed, it holds:*

- a) $\Delta_{ij} + \Delta_{ji} \leq u_{ij}$ for all $\{i, j\} \in E$, and
- b) there exists an edge $\{i, j\} \in E$ such that $\Delta_{ij} + \Delta_{ji} = u_{ij}$.

Lemma 2. *The flow obtained by scaling the final flow by $1/\sigma$ is primal feasible.*

Lemma 3. *Let $\tau = (1 + \epsilon)/\rho$, and denote with L the maximum number of edges in a simple path from a source s^k to one of its sinks $i \in V$. When setting $\delta = \tau(\tau L \max_k \sum_{i \neq s^k} d_i^k)^{-1/\epsilon}$, the final flow scaled by $1/\sigma$ is optimal with a relative error of at most 3ϵ .*

3.3 Implementation Details

Three main practical improvements have been suggested for this kind of approximation algorithms for multicommodity flow problems (see e.g. [11,28,8]): First, it is suggested to choose ϵ more aggressively and to restart the algorithm with a smaller guaranteed error

² We write O^* to denote the “smooth” O-calculus hiding logarithmic factors.

³ A full version of this paper including all proofs can be found in [30].

only if the final performance achieved is not good enough. Of course, this procedure makes sense when we want to compute a solution with a given maximal error as fast as possible. But in our context, we have to select an ϵ that works best for the branch&bound algorithm. We come back to this point in the experiments described in Section 5.

Second, in [8,28] it is suggested to use another length function: Instead of setting $l_{ij} \leftarrow l_{ij}(1 + \epsilon \frac{\Delta}{u_{ij}})$ one can also modify l by $l_{ij} \leftarrow l_{ij} e^{\epsilon \frac{\Delta}{u_{ij}}}$. This modified length function does not change the analysis but it is reported that the algorithm behaves better in practice. Third, it is proposed to use a variable ϕ in the realization of the actual routing tree instead of the fixed ϕ as it is given in the algorithm (in Line 11). This suggestion originates from approximation algorithms prior to [10] that work slightly differently. The idea is to choose ϕ such that a specific potential function is minimized that corresponds to the dual value of the problem. In our algorithm, though, it is not possible to compute ϕ efficiently so that the dual value is minimized. However, it is possible to compute ϕ so that the primal value is maximized, so we experimented with this variant.

Besides these suggestions we present another variation of the approximation algorithm that we call *enhanced scaling*: During the algorithm, for each commodity k , we obtain a flow x^k and possibly also a scalar $\lambda^k = -|x^k|/d_{s^k}^k \geq 0$. In order to construct a feasible flow, instead of scaling all x^k equally, we could also set up another optimization problem to find scalars ξ^k that solve the following LP:

$$\begin{array}{ll} \text{Maximize} & \sum_k \lambda^k \xi^k \\ \text{subject to} & \sum_k \xi^k (x_{ij}^k + x_{ji}^k) \leq u_{ij} \quad \forall \{i, j\} \in E \\ & \xi \geq 0 \end{array}$$

Like that, the bound obtained can be improved in practice. However, this gain has to be paid for by an additional computational effort that, in theory, dominates the overall running time. So the question, how often we use this scaling is a trade off. Experiments have shown that using this scaling after each 100th iteration is a good choice for the instance sizes that we tackle here. Notice that we use this scaling only in order to get a feasible solution value; the primal solution which is used while the algorithm proceeds is not changed! Indeed, we have also tried to continue with the scaled solution in the algorithm, but this variant performs quite badly.

Figure 2 shows the effect of the three different improvements. This example was computed with a DeBruijn graph of dimension 8, and $\epsilon = 0.1$. We have performed many tests with a couple of different settings, so we can say that Figure 2 shows a typical case. It shows the error that results from the difference of the current primal and dual solution depending on the running time. For the application in a branch&bound algorithm, we can stop the calculation as soon as the primal or dual value reaches a specific threshold, so the quality over the whole running time is of interest.

The main result is that the improvement of the enhanced scaling, as it is described above, generally gives the best results over the total run of the algorithm. We also carried out some experiments with all different combinations of the three improvements, but no combination performs as well as the variant with enhanced scaling alone. So in all following experiments we use this variant.

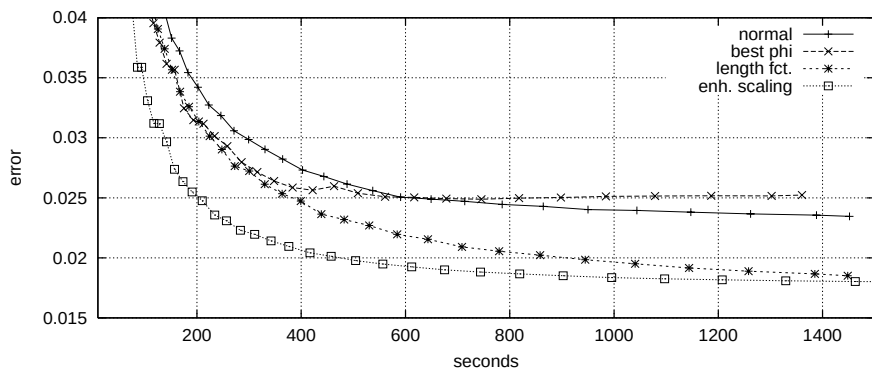


Fig. 2. Comparison of different practical improvements of the FPTAS.

4 A Branch&Bound Algorithm

Our main goal is the computation of exact solutions for graph bisection problems. Thus, we construct a branch&bound algorithm using the described VarMC-bound as lower bound for the problems. A detailed description of the implementation can be found in [31]. In the following, we give a brief survey on the main ideas: First, we heuristically compute a graph bisection using *PARTY* [26]. Since this start-solution is optimal in most cases, we only have to prove optimality. We use a pure depth first search tree traversal for this purpose. The branching is done on the decision whether two specific vertices $\{v, w\}$ stay in the same partition (join) or if they are separated (split). A join is performed by merging these two vertices into one vertex. A split is performed by introducing an additional commodity from vertex v to vertex w whose entire amount is known to cross the cut. Thus, it can be added to the *CutFlow* completely. The selection of the pair $\{v, w\}$ for the next branching is done with the help of an upper bound on the lower bound. Additionally to this idea described in [31], the selection is restricted to pairs $\{v, w\}$ (if any) where one node (say, v) has been split with some other node $u \neq w$ before. Then, split of $\{v, w\}$ implies a join of $\{u, w\}$, of course.

The “Forcing Moves” strategy described in [31], Lemma 2, is strengthened. Forcing moves are joins of vertices that can be deduced in an inference process. In the original version, only adjacent vertices $\{v, w\}$ were considered. Now, we look at a residual graph with edge-capacities corresponding to the amount of capacity which is not used by a given VarMC solution. Two vertices v and w can be joined if the maximal flow from v to w in the residual graph exceeds a specific value.

5 Numerical Results

We now present the results of our computational experiments. All of them were executed on systems with Intel Pentium-III, 850 MHz, and 512 MByte memory. To show the behavior on different kinds of graphs, we use four different sets of 20 randomly generated graphs: The set *RandPlan* contains random maximal planar graphs with 100 vertices;

the graphs are generated using the same principle as it is used in LEDA [23]. Benchmark set *RandReg* consists of random regular graphs with 100 vertices and degree four; for its generation, the algorithm from Steger and Wormald [33] is used. The set *Random* contains graphs with 44 vertices where every pair $\{v, w\}$ is adjacent with probability 0.2. The set *RandW* consists of complete graphs with 24 vertices where every edge has a random weight in the interval $\{0, \dots, 99\}$. In most works on exact graph partitioning (see e.g. [2,18]), sets like *Random* and *RandW* are used for the experiments. We added the sets of random regular and random planar graphs here, because we believe that more structured graph classes should also be considered with respect to their bigger relevance for practical applications.

5.1 Lower Bounds Using the FPTAS

The best choice of ϵ for the use in a branch&bound environment is a trade-off. If ϵ is too big, the bound approximation computed is too bad, and the number of subproblems in the search-tree explodes. On the other hand, if ϵ is chosen too small, the bound approximation for each subproblem is too time consuming.

graph		$\epsilon = 0.025$		$\epsilon = 0.05$		$\epsilon = 0.1$		$\epsilon = 0.25$		$\epsilon = 0.5$		$\epsilon = 0.75$	
		time subp.		time subp.		time subp.		time subp.		time subp.		time subp.	
(1)	RandPlan	6395	461	2158	461	962	463	587	557	450	1466	562	5273
	RandReg	2192	22	1107	23	561	26	196	37	126	90	163	489
	Random	2620	113	525	114	228	118	98	139	80	280	186	2188
	RandW	1383	37	472	46	176	62	76	150	85	788	1289	3859
(2)	RandPlan	3013	412	1009	406	587	381	133	239	54	117	35	78
	RandReg	2186	22	1083	23	622	25	181	34	117	67	160	173
	Random	2249	107	465	108	209	111	83	121	49	164	47	328
	RandW	737	14	283	17	122	25	44	54	35	188	41	582

Fig. 3. Times and sizes of the search-trees of the branch&bound algorithm using the approximation algorithm with different ϵ 's. (1): without forcing moves, (2): with forcing moves.

graph	CPLEX		Approx.		Cost-Dec.	
	time subp.		time subp.		time subp.	
RandPlan	74	7	54	117	889	26
RandReg	993	21	117	67	557	28
Random	350	99	49	164	612	106
RandW	30	9	35	188	120	11

Fig. 4. Average running times (seconds) and sizes of the search-trees using the different methods for computing the VarMC-bound.

Figure 3 shows the resulting running times and the number of subproblems of the branch&bound algorithm using approximated bounds. The results are the averages on all 20 instances for every set of graphs.

The results without forcing moves show the expected behavior for the choice of ϵ . The smaller ϵ is, the smaller is the number of search nodes. This rule is not strict when using forcing moves: looking at the random planar graphs, we see that less good solutions of the VarMC bound can result in stronger forcing moves so that the number of subproblems may even decrease. The figure also shows that the effects of forcing moves are different for the different classes of graphs and also for changing ϵ 's. Altogether, the experiments show that setting ϵ to 0.5 only is favorable, which is a surprisingly large value.

5.2 Comparison of Lower Bound Algorithms

We give a comparison of the results of the branch&bound algorithm with forcing moves using the following three different methods for computing the VarMC-bound: 1. standard barrier LP-solver (CPLEX, Version 7.0, primal and dual simplex algorithms cannot compete with the three approaches presented here), 2. the approximation algorithm with $\epsilon = 0.5$ and enhanced scaling, and 3. a Lagrangian relaxation based cost-decomposition routine using a max-cutflow formulation and the Crowder rule in the subgradient update (see [30] for details). Figure 4 shows the results on the four different benchmark sets. In general, the approximation algorithm with the enhanced scaling gives the best running times, even though the search-trees are largest.

Graph	$ V $ $ E $		bw	CPLEX		Decomp.		Approx.		CUTSDP	
				Bound	Time	Bound	Time	Bound	Time	Bound	Time
Grid 10x11	110	199	11	11.00	16	11.00	54	10.59	2	8.48	30
Torus 10x11	110	220	22	20.17	15	20.17	48	19.66	2	17.63	30
SE 8	256	381	28	26.15	484	25.69	244	24.94	16	18.14	453
DB 8	256	509	54	49.54	652	49.05	322	46.95	21	35.08	443
BCR m8	148	265	7	7.00	22	7.00	64	6.96	5	5.98	80
ex36a	36	297	117	104.17	9	104.16	26	97.57	<1	116.41	1

Fig. 5. Comparison of the different VarMC bounds with the semi-definite bound. The bound quality at the root node as well as the time for its computation are given.

Note that, with respect to the memory consumption, approximation and cost-decomposition are preferable to the barrier algorithm: When the graphs we consider become larger, for DeBruijn 9 or Shuffle-Exchange 9, for example, 2 GB main memory are not enough to allow the application of CPLEX. Thus, it cannot be applied to compute the corresponding bisection widths, whereas both cost-decomposition and approximation allowed us to prove a bisection width of 92 and 48, respectively. Apart from being more memory efficient, we observe that cost-decomposition does not allow us to solve our generalized maximum multicommodity flow problem faster than standard LP methods.

5.3 Comparison with Semi-definite Lower Bounds

Finally, we present a comparison of the different methods for computing the VarMC-Bound with a lower bound on the graph bisection problem based on semi-definite programming presented in [18]. The work in [18] is the most recent and successful on the graph bisection problem apart from the VarMC-bound. So we compare our lower bounds with these ones. In order to compute the semi-definite bounds we have used the CUTSDP-package, which is publicly available. The program uses parameters *maxlarge* and *maxsmall*. According to the setting in [18], we set *maxlarge* := 1, and *maxsmall* := 10.

The results are presented in Figure 5. Here we have not used randomly generated graphs but more structured graphs: a grid, a torus, the shuffle-exchange graph of dimension 8, and the DeBruijn graph of dimension 8. Additionally we report the results on the “BCR m8” graph which stems from a finite elements application, and the “ex36a” graph which is a small random dense graph introduced in [18].

The results show the power of the approximation algorithm in the application of graph bisection. Its bounds are nearly as good as the optimal bounds of the VarMC method computed with CPLEX and much better than the semi-definite bounds. And the running times are clearly the smallest ones out of the four compared methods. The results also show that the semi-definite bound is better for the very dense random graph while the VarMC bound is better for more structured or sparse graphs.

6 Conclusions

We developed an algorithm for the approximation of the VarMC-bound on the bisection width of a graph. It is based on an approximation scheme for maximum multicommodity flows and yields an ϵ -approximation in time $O^*(m^2/\epsilon^2)$. We experimented with different practical improvements that have been suggested for this kind of multicommodity flow approximation scheme and were able to improve the practical behavior by using the new enhanced scaling technique.

When comparing the approximation algorithms with a barrier LP-solver and a Lagrangian relaxation based cost-decomposition algorithm, we found that it is favorable to use the approximation scheme, both with respect to the memory consumption and running time. The VarMC-approximation scheme allowed us to compute the bisection width of large graphs, such as DeBruijn 9 (the bisection width is 92), Shuffle-Exchange 9 (48), and Shuffle-Exchange 10 (82), which were unknown and out of the reach of exact graph bisection algorithms before.

References

1. C. Albrecht. Provably good global routing by a new approximation algorithm for multicommodity flow. In *Proc. of the Int. Conference on Physical Design*, pages 19–25, 2000.
2. L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78:243–263, 1997.
3. T.G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112:73–99, 2001.

4. J. Farvolden, K. Jones, I. Lustig, and W. Powell. Multicommodity Network Flows — The Impact Of Formulation On Decomposition. *Mathematical Programming*, 62:95–117, 1993.
5. J. Farvolden, W. Powell, and I. Lustig. A primal partitioning solution for the arc-chain formulation of a multicommodity network flow problem. *Operations Research*, 41(4):669–693, 1993.
6. C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: a computational study. *Mathematical Programming*, 81:229–256, 1998.
7. L. Fleischer and K.D. Wayne. Fast and simple approximation schemes for generalized flow. In *Proceedings 10th Symposium on Discrete Algorithms*, 1999.
8. L. K. Fleischer. Approximating Fractional Multicommodity Flow Independent of the Number of Commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.
9. A. Frangioni. A Bundle type Dual-ascent Approach to Linear Multi-Commodity Min Cost Flow Problems. Technical Report TR-96-01, Dipartimento di Informatica, Università di Pisa, 1996.
10. N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.
11. A.V. Goldberg, J.D. Oldham, S.A. Plotkin, and C. Stein. An Implementation of a Combinatorial Approximation Algorithm for Minimum-Cost Multicommodity Flow. In *Integer Programming and Combinatorial Optimization*, volume 1412, pages 338–352, 1998.
12. M.D. Grigoriadis and L.G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4:86–107, 1994.
13. M.D. Grigoriadis and L.G. Khachiyan. Approximate minimum-cost multicommodity flows. *Math. Programming*, 75:477–482, 1996.
14. B. Hendrickson and B. Leland. The chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, 1994.
15. E. Johnson, A. Mehrotra, and G. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993.
16. G. Karakostas. Fast Approximation Schemes for Fractional Multicommodity Flow Problems. In *SIAM Symposium on Discrete Algorithms*, 2002.
17. D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proc. of the 27th Symp. on Theory of Computing*, pages 18–25, 1995.
18. S. E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3):177–191, 2000.
19. G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical Report TR 98-019, Dept. of Computer Science, Univ. of Minnesota, 1998.
20. P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23:466–487, 1994.
21. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufman, 1992.
22. T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast Approximation Algorithms for Multicommodity Flow Problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.
23. K. Mehlhorn and S. Nähler. LEDA: A Platform for Combinatorial and Geometric Computing. *Communications of the ACM*, 38(1):96–102, 1995.
24. F. Pellegrini and J. Roman. SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. HPCN'96*, pages 493–498, 1996.
25. S.A. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. of Operations Research*, 20:257–301, 1995.

26. R. Preis and R. Diekmann. PARTY - A Software Library for Graph Partitioning. In B.H.V. Topping, editor, *Advances in Computational Mechanics with Parallel and Distributed Processing*, pages 63–71. Civil-Comp Press, 1997.
27. T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In *Proc. of the 6th Symp. on Discrete Algorithms*, pages 486–492, 1995.
28. T. Radzik. Experimental study of a solution method for multicommodity flow problems. In *Proc. of the 2. Workshop on Algorithm Engineering and Experiments*, pages 79–102, 2000.
29. M. Sato. Efficient implementation of an approximation algorithm for multicommodity flows. Master's thesis, Graduate School of Engineering Science, Osaka University, 2000.
30. M. Sellmann. *Reduction Techniques in Constraint Programming and Combinatorial Optimization*. PhD thesis, University of Paderborn, Germany, <http://www.upb.de/cs/sello/diss.ps>, 2002.
31. N. Sensen. Lower Bounds and Exact Algorithms for the Graph Partitioning Problem. In F. Meyer auf der Heide, editor, *Algorithms - ESA 2001*, volume LNCS 2161, pages 391–403, 2001.
32. F. Shahrokhi and D.W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318–334, 1990.
33. A. Steger and N.C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probab. and Comput.*, 8:377–396, 1999.
34. N. Young. Randomized rounding without solving the linear program. In *Proc. of the 6th Symp. on Discrete Algorithms*, pages 170–178, 1995.

A Linear Time Heuristic for the Branch-Decomposition of Planar Graphs

Hisao Tamaki*

Meiji University, Kawasaki 214, Japan,
tamaki@meiji.ac.jp

Abstract. Let G be a biconnected planar graph given together with its planar drawing. Let $\text{VF}(G)$ denote the bipartite graph on the sets of vertices and of faces of G such that each of its edges represents an incidence in G between a face and a vertex. Let α_G denote the maximum distance in $\text{VF}(G)$ from the outerface of G . We show that there always exists a branch-decomposition of G with width α_G and that such a decomposition can be constructed in linear time. We also give experimental results, in which we compare the width of our decomposition with the optimal width and with the width obtained by some heuristics for general graphs proposed by previous researchers, on test instances used by those researchers. On 56 out of the total 59 test instances, our method gives a decomposition within additive 2 of the optimum width and on 33 instances it achieves the optimum width.

1 Introduction

Let G be a graph with vertex set $V(G)$ and edge set $E(V)$. A *branch-decomposition* [6] of G is a ternary tree (i.e., a tree in which each non-leaf node has degree exactly 3), in which each edge of G appears as a leaf exactly once. To avoid confusion, we will speak of nodes of branch-decompositions, while we speak of vertices of graph G .

Let B be a branch-decomposition of G . For each ordered pair (p, q) of adjacent nodes in B , we denote by $B_{p,q}$ the subtree of B induced by the nodes reachable from p , including p , without using the edge between p and q . For each subtree B' of B , let $L(B')$ denote the set of edges of G contained, as leaves, in B' . Then, $\{L(B_{p,q}), L(B_{q,p})\}$ is a *separation* of $E(G)$, i.e., $L(B_{p,q}) \cap L(B_{q,p}) = \emptyset$ and $L(B_{p,q}) \cup L(B_{q,p}) = E(G)$. We say that the edge between p and q *induces* this separation. The *width* of a separation $\{S, E(G) \setminus S\}$ of $E(G)$ is the number of vertices of G that are incident to some edge of S and at the same time incident to some edge of $E(G) \setminus S$. The width of a branch-decomposition is the maximum width of the separation induced by an edge of the decomposition. Finally, the branch-width of G is the minimum width over all branch-decompositions of G .

Given a branch-decomposition of G , dynamic programming algorithms for numerous problems can be designed that run in time linear in the size of the

* This work was done while the author was visiting Max-Planck-Institut für Informatik, Saarbrücken, Germany.

graph and exponential in the width of the decomposition. These problems include, to name a few, coloring, vertex cover, Steiner tree and the traveling salesman problem. Indeed, the branch-width of a graph G is linearly related to the tree-width [5] of G and therefore all the problems solvable in linear time on graphs with bounded tree-width (see [1,2] for example) can be solvable in linear time on bounded branch-width graphs. Therefore, it is of extreme interest to recognize graphs of small branch-width and to find a branch-decomposition of small width for a given graph. Though the problem of deciding the width of a given general graph is NP-complete, it can be solved in $O(n^2)$ time for planar graphs by the algorithm of Seymour and Thomas [7]. Their algorithm can also be used to find a branch-decomposition with the optimal width, but the running time for this task is $O(n^4)$. Therefore, for large graphs, we still need to resort to heuristics even if the graph is planar.

Two effective heuristics for the branch-decomposition of general graphs are known. One is called the eigenvector method due to Cook and Seymour [3] and the other the diameter method due to Hicks [4]. Both methods proceed recursively, building a binary tree of decomposition (which can in the end be viewed as a ternary tree disregarding the root). At each step of recursion, we have a set of edges E and try to decompose this set in two sets E_1 and E_2 so that the widths of the separation $\{E_1, E(G) \setminus E_1\}$ and of $\{E_2, E(G) \setminus E_2\}$ are both small. We also wish to avoid a decomposition that would result in a large width in the further recursive steps. The two methods differ in the manner they try to achieve this second goal. As their names suggest, one uses the eigenvectors of the adjacency matrix, and the other the diameter, of the local graph induced by E to guide the separation step. Though these heuristics have been successful to some extent, the running time observed by experiments has a rather high complexity with respect to the input size and no theoretical upper bound is known.

In this paper, we introduce a heuristic specialized to planar instances. This heuristic is completely different from the above general heuristics and crucially depends on the planarity of the instance. The resulting algorithm is extremely simple and the running time is linear in the size of the input.

Formally, the following theorem captures our heuristic. Let G be a planar graph drawn in the plane and let $F(G)$ denote the sets of its faces. Let $\text{VF}(G)$ denote the bipartite graph on $V(G)$ and $F(G)$ with each edge of $\text{VF}(G)$ representing an incidence in G . Let α_G denote the maximum of the distance in $\text{VF}(G)$ from the outer face of G to a face or vertex x , over all $x \in V(G) \cup F(G)$.

Theorem 1. *Given a biconnected planar graph G with a planar drawing, a branch-decomposition of G with width at most α_G can be constructed in linear time.*

The width of the decomposition we obtain from this theorem clearly depends on the choice of the outer face. For some geometric instances, it appears that the outer face in the natural drawing is usually the best choice for minimizing α_G . In general, we may need to look for a good choice of the outer face at the cost of increased running time.

The experimental results summarized in Section 5 suggest that our heuristic exploits planarity extremely well in finding good branch-decompositions.

The rest of this paper is organized as follows. In the next section, we describe the notion of carving-decompositions of a graph, which is closely related to that of branch-decompositions, and state a theorem on carving-decompositions which implies our main theorem. In Section 3, we introduce a certain type of spanning trees of a planar graph, which we call medial-axis trees, and study its properties. In Section 4, we prove the theorem on carving-decompositions using a construction based on a medial-axis tree. Section 5 summarizes some experimental results reported in a full-version of this paper. We mention some applications and future research directions in the conclusion section.

2 Carving- and Branch-Decompositions

In this section, we describe the notion of carving-decompositions and state their relation to branch-decompositions. Our theorem on branch-decompositions stated in the introduction is a consequence of another theorem on carving-decompositions stated in this section.

Let G be a graph. A *carving-decomposition* [6] of G is defined similarly to a branch-decomposition, but differs in that it is a decomposition of $V(G)$ rather than of $E(G)$. More formally, a *carving-decomposition* of G is a ternary tree in which each vertex of G appears as a leaf exactly once. Let C be a carving-decomposition of G . For each ordered pair (p, q) of adjacent nodes in B , we denote by $C_{p,q}$ the subtree of C induced by the nodes reachable from p , including p , without using the edge between p and q . For each subtree C' of C , let $L(C')$ denote the set of vertices of G contained, as leaves, in C' . We say that the edge between p and q induces the separation $\{L(C_{p,q}), L(C_{q,p})\}$ of $V(G)$. We also say that this separation is *in* C . The *cut* associated with a separation $\{S, V(G) \setminus S\}$ of $V(G)$ is the set of edges each having one end in S and the other end in $V(G) \setminus S$. The *width* of a separation of $V(G)$ is the cardinality of its associated cut. The width of a carving-decomposition is the maximum width over all the separations in the decomposition. The *carving-width* of G is the minimum width over all the possible carving-decompositions of G .

In the rest of this paper, we are only concerned with planar graphs. When we speak of a planar graph G , we assume that a planar drawing of G is given together with G . When we consider the branch-decomposition of G , we will assume that G is biconnected, since otherwise the branch-decomposition of G can be obtained by combining the branch-decompositions of the biconnected components of G . We will moreover assume that G does not contain parallel edges, as they do not affect the branch-width. However, we will need to consider graphs with parallel edges when we consider carving-decompositions, especially in the following context of relating the branch- and carving-decompositions. We will always assume that our graphs do not contain self-loops.

Let G be a planar graph. The *medial graph* [7] of G , denoted by $M(G)$ is constructed as follows. $M(G)$ has a distinct vertex x_e for each edge e of G . It has

an edge between x_e and $x_{e'}$ if and only if e and e' share an endvertex v and they are consecutive in the counterclockwise order around v . When e and e' are the only edges incident with v , then there are two parallel edges between x_e and $x_{e'}$. It is not difficult to see that $M(G)$ is planar: draw each x_e at the bisection point of the drawing of e and for each vertex v of G , with edges $e_1, \dots, e_k$ incident with it in counterclockwise order, draw a closed curve around v going through $x_{e_1}, \dots, x_{e_k}$ in this order. See Figure 2 for an example. Note that there are two types of faces of $M(G)$: one type corresponds to a vertex of G and the other corresponds to a face of G . In Figure 2, the faces corresponding to vertices of G are shaded. Note that the dual of $M(G)$ is isomorphic to $\text{VF}(G)$ and hence is bipartite.

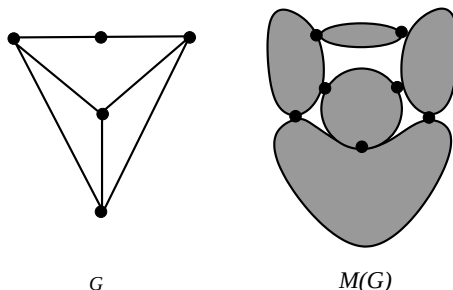


Fig. 1. Planar graph G and its medial graph

Seymour and Thomas [7] show that the carving-width of $M(G)$ is exactly twice the branch-width of G . We only need the following lemma to this theorem.

Lemma 1 (Seymour and Thomas [7]). *Let C be a carving-decomposition of $M(G)$ with width w . Then G has a branch-decomposition of G with width at most $w/2$.*

Our main theorem is obtained by applying the following theorem on carving-decomposition to the medial graph.

Let G be a planar graph. We denote by G^* the planar dual of G . To each vertex v , edge e , or face f of G corresponds a face v^* , an edge e^* , or a vertex f^* of G^* , called duals of v , e and f , respectively. For each vertex or face x of G and each edge e of G , x is incident with e^* in G^* if and only if x is incident with e in G .

For each face f of G , the *height* of f , denoted by $h(f)$, is defined to be the length of the shortest path in G^* from f_{out}^* to f^* , where f_{out} is the outer face of G and the length of a path is the number of edges in the path.

Let v be a vertex of G . For two distinct faces f, g incident to v , let $\pi_v^*(f, g)$ denote the dual path from f^* to g^* counterclockwise around the dual face v^* . Define $\beta_v(f, g)$ by $\beta_v(f, g) = |\pi_v^*(f, g)| + h(f) + h(g)$, where $|\cdot|$ denotes the length of the path. Then define β_v for each $v \in V(G)$ by:

$$\beta_v = \min_{f \neq g} \{ \max \{ \beta_v(f, g), \beta_v(g, f) \} \},$$

where the minimum is taken over all distinct pairs f, g of faces incident with v . Finally, we define β_G to be the maximum of β_v over all $v \in V(G)$. We also denote by δ_G the maximum vertex degree of G .

Theorem 2. *Given a planar graph G , a carving-decomposition of G with width at most $\max\{\beta_G, \delta_G\}$ can be constructed in linear time.*

We need the following observation in order to obtain Theorem 1 from this theorem.

Proposition 1. *Let G be a biconnected planar graph and let $M(G)$ be the medial graph of G . Then $\beta_{M(G)} \leq 2\alpha_G$.*

Proof. Let v be an arbitrary vertex of $M(G)$. Note that every vertex of $M(G)$ has degree 4. Let f_1, f_2, f_3, f_4 be the faces incident with v in counterclockwise order and, without loss of generality, assume that $h(f_1)$ is the largest of $h(f_i)$, $1 \leq i \leq 4$. Since the dual of $M(G)$ is bipartite, two adjacent faces cannot have the same height and hence we have $h(f_2) = h(f_4) = h(f_1) - 1$. Therefore, we have $\beta_v(f_2, f_4) = \beta_v(f_4, f_2) = 2(h(f_1) - 1) + 2 = 2h(f_1)$ and hence $\beta_v \leq 2h(f_1)$. But $h(f_1)$ equals the distance in $VF(G)$ from the outer face to the face or vertex of G corresponding to f_1 . It follows that we have $\beta_v \leq 2\alpha_G$ for every vertex v of $M(G)$.

Since any carving-decomposition of $M(G)$ with width $2w$ straightforwardly translates, in linear time, into a branch decomposition of G with width w through Lemma 1, Theorem 1 is an immediate consequence of Theorem 2 and Proposition 1.

We prove Theorem 2 in the following two sections.

3 Medial-Axis Tree

Let G be a planar graph. Throughout this and the next sections, we assume that G is 2-edge-connected. Note that this is true for medial graphs. In general, this assumption is not essentially restrictive either, since the problem of finding a good carving decomposition of general planar graph G trivially reduces to that of finding good carving decompositions of the 2-edge-connected components of G .

Our carving-decomposition of G is based on a certain spanning tree of G which we call a *medial-axis tree* of G . Let f_{out} denote the outer face of G , and let T^* be a breadth-first tree in G^* , the dual of G , rooted at f_{out}^* . That is, each path in T^* from the root to a vertex f^* is the shortest path from f_{out}^* to f^* in G^* .

Let T be the spanning tree of G dual to T^* , i.e., T is a spanning subgraph of G such that an edge e of G is in T if and only if e^* is not in T^* . Since the connectedness of T is equivalent to the acyclicity of T^* and vice versa, T

is indeed a spanning tree of G . We call T a medial-axis tree of G . This name is in analogy of the geometric notion: the *medial-axis* of a bounded region in the plane is the locus of points p such that there are at least two points on the boundary of the region that minimize the distance to p over all the points on the boundary. The property of each edge e of T that the two faces incident with e are almost equidistant in the dual graph to the outer face (the distances differing by at most 1 and the shortest dual path for neither face crossing e) supports this analogy, though somewhat in a loose manner. Given a drawing of G , it is clear that a medial-axis tree of G can be constructed in linear time.

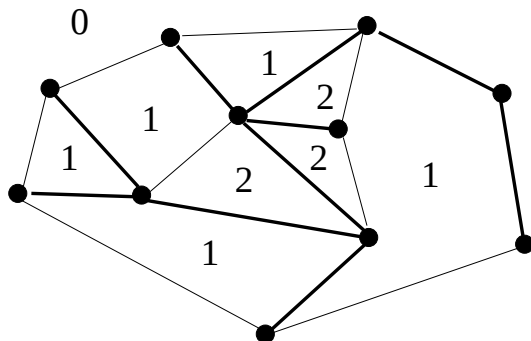


Fig. 2. A medial-axis tree of a planar graph: bold lines indicate tree edges and the number in each face indicates its height

Fix a medial-axis tree T of G and let v be a vertex of G . Let $e_1, \dots, e_k$ of T be the edges incident to v listed in the counterclockwise order. Between edges e_i and e_{i+1} (the latter should be interpreted as e_1 when $i = k$), v may have more than one incident face but the face with the minimum height is unique, since the dual of these faces belong to a path of T^* and there is a unique vertex of this dual path closest to the root. We denote this face of minimum height between e_i and e_{i+1} by f_i and call $f_1, \dots, f_k$ the *delimiting faces* of v .

For each distinct pair f, g of delimiting faces of v , let $E_v(f, g)$ denote the set of edges of T incident to v that appear counterclockwise after f and before g around v . In the notation of the previous paragraph, $E_v(f_i, f_j) = \{e_h \mid i < h \leq j\}$ if $i < j$.

For each vertex v of G and a distinct pair f, g of its delimiting faces, let $T_v(f, g)$ denote the subtree of T consisting of v and all vertices reachable from v through a path of T containing some edge of $E_v(f, g)$. We call such a subtree a *contiguous subtree* of T at v specified by f and g .

We are interested in analyzing the separations of the form $\{V(T'), V(G) \setminus V(T')\}$, where T' is a contiguous subtree of T , since they appear in the carving-decomposition we are going to construct.

Let f and g be distinct delimiting faces of v . Let $\lambda_v^*(f, g)$ denote the path in T^* that goes from f^* up to the lowest common ancestor of f^* and g^* and then

down to g^* . Draw a closed curve γ through v that stays within the faces of G corresponding to the vertices of $\lambda_v^*(f, g)$ and crosses, exactly once, each edge of G corresponding to an edge of $\lambda_v^*(f, g)$. Observe that this curve separates the subtree $T_v(f, g)$ from the other parts of T . More precisely, any edge of G between a vertex of $T_v(f, g)$ and a vertex not in $T_v(f, g)$ must either be incident with v or cross γ . Edges in the first category correspond to the dual edges in $\pi_v^*(g, f)$ in the notation of Section 2. Edges in the second category correspond to the edges in $\lambda_v^*(f, g)$. Since the length of $\lambda_v^*(f, g)$ is at most $h(f) + h(g)$, we have the following lemma.

Lemma 2. *The width of separation $\{V(T_v(f, g)), V(G) \setminus V(T_v(f, g))\}$ in G is at most $\beta_v(g, f)$.*

4 Constructing the Carving-Decomposition

We are now ready to describe the construction of our carving-decomposition that satisfies the claim of Theorem 2.

Let T be a medial-axis tree of the given planar graph G . We first construct a local ternary tree C_v for each v . These local trees are later combined into a carving-decomposition of G .

Fix vertex v of G and let k be its degree in T . Let $u_1, \dots, u_k$ be the neighbors of v in T , listed in the counterclockwise order. C_v is a ternary tree with $k + 1$ leaves, which are labeled by $v, u_1, \dots, u_k$, constructed as follows.

First suppose v is a leaf of T , i.e., $k = 1$. Then, in this case, C_v is a ternary tree with two leaves and with no internal nodes, one leaf being labeled with v and the other with u_1 .

Now suppose $k \geq 2$. Let f, g be a pair of distinct delimiting faces of v that minimizes $\max\{\beta_v(f, g), \beta_v(g, f)\}$. We note that this pair indeed minimizes this quantity over all pairs of faces incident with v , and hence $\max\{\beta_v(f, g), \beta_v(g, f)\} \leq \beta_v$. By renumbering, suppose $E_v(f, g) = \{(v, u_i) \mid 1 \leq i \leq j\}$ and $E_v(g, f) = \{(v, u_i) \mid j < i \leq k\}$ where $1 \leq j < k$. Our local ternary tree C_v is constructed as follows. We start from a ternary tree with three leaves l_1, l_2, l_3 , in the counterclockwise order, and one internal node. We label l_1 with v . We extend the tree from l_2 in an arbitrary manner, as long as it remains ternary, until the number of leaves in this extended part is j and label the resulting leaves with $u_1, \dots, u_j$ in the counterclockwise order. Similarly, we extend the tree from l_3 until the number of leaves in this part is $k - j$ and label the resulting leaves with $u_{j+1}, \dots, u_k$ in the counterclockwise order. This completes the description of C_v . See Figure 4 for an example.

Having constructed the local tree C_v for each vertex v of G , we now combine them into one tree. Let vertices u, v be adjacent in T . Then, we identify the leaf of C_v labeled u with the leaf of C_u labeled v and then turn the two incident edges into one, removing this identified node. Doing this to all pairs u, v of vertices adjacent in T , we obtain a ternary tree C whose leaves are labeled with the vertices of G : this is our carving-decomposition of G .

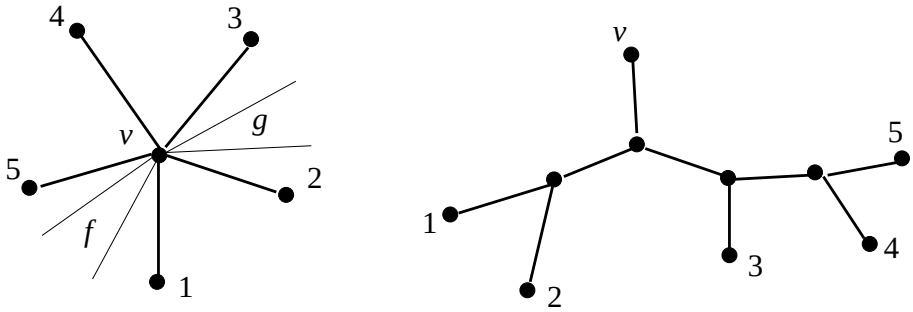


Fig. 3. Vertex v with its tree edges (left) and its local graph C_v (right)

It remains to show that the carving-decomposition C thus constructed has width at most $\max\{\beta_G, \delta_G\}$. Consider an arbitrary edge (p, q) of C . If either p or q is a leaf, labeled with $v \in V(G)$, then the width of the separation induced by this edge is the degree of v and therefore at most δ_G . Suppose now that neither p nor q is a leaf. Let v be the vertex of G such that edge (p, q) comes from the local tree C_v . When the choice of v is not unique (which happens when the edge (p, q) is the result of concatenating two edges from two local trees), we choose an arbitrary one of the two possibilities. By abuse of notation, call the corresponding edge in the local tree also (p, q) . Let U be the set of labels of the leaves of C_v that lie in the same side of the leaf labeled with v viewed from edge (p, q) . Since $U \setminus \{v\}$ is a set of neighbors of v in T that appear contiguously around v in the counterclockwise order, $E_U = \{(v, u) \mid u \in U \setminus \{v\}\}$ can be represented as $E_v(f', g')$ for some delimiting faces f', g' of v . Moreover the construction of C_v guarantees that, this set $E_v(f', g')$ is a superset of either $E_v(f, g)$ or $E_v(g, f)$, where (f, g) is the pair of delimiting faces of v chosen to minimize $\max\{\beta_v(f, g), \beta_v(g, f)\}$ in the construction of C_v . This implies that $\beta_v(g', f')$ is at most $\max\{\beta_v(f, g), \beta_v(g, f)\} \leq \beta_G$. The separation induced by edge (p, q) of C is $\{V(T_v(f', g')), V(G) \setminus V(T_v(f', g'))\}$ and by Lemma 2, its width is at most $\beta_v(g', f')$. We conclude that the width of each separation in C is at most β_G .

We show that the time required to construct C is linear in the size of G . It suffices to show that the local tree at each v can be constructed in time proportional to the degree of v . The only non-trivial part is the determination of a pair of delimiting faces f, g of v that minimizes $\max\{\beta_v(f, g), \beta_v(g, f)\}$.

For the purpose of analysis, it is easier to consider the problem of finding a pair f, g of any faces incident to v , which is not necessarily a pair of delimiting faces, that minimizes $\max\{\beta_v(f, g), \beta_v(g, f)\}$. These problems are essentially equivalent, because if we find an optimal pair f, g for the latter problem then it can straightforwardly be converted into a pair of delimiting faces f', g' such that $\max\{\beta_v(f, g), \beta_v(g, f)\} = \max\{\beta_v(f', g'), \beta_v(g', f')\}$.

Fix an arbitrary face f incident to v and move face g counterclockwise around v . When g advances one step, one of the following three happens: (1) $\beta_v(f, g)$

increases by one and $\beta_v(g, f)$ decreases by one (when $h(g)$ does not change); (2) $\beta_v(f, g)$ increases by two and $\beta_v(g, f)$ does not change (when $h(g)$ increases); (3) $\beta_v(f, g)$ does not change and $\beta_v(g, f)$ decreases by two (when $h(g)$ decreases). In any case, $\beta_v(f, g) - \beta_v(g, f)$ increases by two. Therefore, among the faces g that minimize $\max\{\beta_v(f, g), \beta_v(g, f)\}$, there is one for which $|\beta_v(f, g) - \beta_v(g, f)| \leq 1$ holds. Call this face g_f . In case of a tie, we choose the one satisfying $\beta_v(f, g_f) > \beta_v(g_f, f)$, for concreteness.

Let f' be the face immediately after f in the counterclockwise order around v . Since $\beta_v(f', g_f) - \beta_v(g_f, f') = \beta_v(f, g_f) - \beta_v(g_f, f) - 2$ and $|\beta_v(f, g_f) - \beta_v(g_f, f)| \leq 1$, we have $\beta_v(f', g_f) < \beta_v(g_f, f')$ and hence $g_{f'}$ cannot lie after f' and strictly before g_f counterclockwise. Therefore, to find a pair f, g_f to minimize $\max\{\beta_v(f, g_f), \beta_v(g_f, f)\}$, we only need two pointers, one for f and the other for g_f , each moving counterclockwise. Neither of these pointers needs to move more than twice around v and therefore our task can be performed in time proportional to the degree of v .

5 Experiments

We implemented the heuristic and compared its performance with the branch-decomposition heuristics for general graphs due to Cook and Seymour [3] and to Hicks [4] on the planar test instances used by these researchers, which are available at [12].

These are Delaunay triangulations of point sets taken from TSPLIB [11], the number of points ranging from 100 to 2319 and the optimal width ranging from 7 to 44. Experimental data of their methods on these instances, as well as their optimum width, are taken from [4].

Table 1 summarizes the quality of decompositions obtained by each method. In this table, “Medial-axis” is our heuristic for planar graphs, “diameter” is the diameter method of Hicks, “eigenvector” is the eigenvector method of Cook and Seymour and “hybrid” is a hybrid of the diameter and eigenvector methods due also to Hicks. For each of the four methods, the number of instances (out of the total 59) on which the method achieves the width of optimum plus i is shown for each $0 \leq i \leq 4$.

The quality of the decompositions for the first three methods as listed in this table are quite similar, while the eigenvector method gives somewhat inferior results.

On the other hand, our method has a great advantage in the computation time. For example, on an instance called ch130 (with 130 vertices and width 10), our method takes 5 milliseconds while the other three methods take 2, 11, and 4 seconds in the listing order of Table 1. The difference tends to be larger for instances with larger width: on an instance called nrw1319 (with 1319 vertices and width 31), for example, our method takes 9 milliseconds while the other three takes 8907, 8456, and 1216 seconds. See [8] for more details. To summarize, our method is typically 10^3 times faster than all the other three methods, and in some cases 10^6 times faster than some of the other methods. Although the CPU

used in our experiment (900 MHz Ultra SPARC-III) is considerably faster than the one used for the other methods(143MHz SPARC1), it may be concluded that our method takes advantage of the planarity extremely well.

Table 1. Quality of the decompositions: summary

	Medial-axis	Diameter	Hybrid	Eigenvector
opt	33	34	35	23
opt + 1	16	15	15	12
opt + 2	7	3	7	4
opt + 3	2	4	0	3
opt + 4	1	0	1	6
greater	0	3	1	11

6 Concluding Remarks

Although our main result is formulated for branch-decompositions, the underlying theorem on carving decompositions (Theorem 2) is equally important in its own right, because for some problems the carving-decomposition is more suitable than the branch-decomposition as the framework for dynamic programming. In fact, the carving-decomposition heuristic given by Theorem 2 is one of the core components in our TSP solver for sparse planar graphs [9], which has recently discovered new best tours for two unsolved TSPLIB instances brd14051 and dl8512 [10]. This solver takes the approach similar to that of Cook and Seymour [3] who use branch-decomposition to solve TSP on small-width graphs that arise from the union of near-optimal tours. The advantage of our solver comes in large part from the heuristic described in this paper which enables the quick recognition of small-width graphs and gives high quality decompositions.

We envision applications of our fast branch- and carving-decomposition methods to other areas, such as various optimization problems on surface meshes which occur in graphics and geometric modeling context. The approach would consist of repeated improvements, in each step of which we take a submesh, whose width as a planar graph is small enough, and compute an optimal local improvement on the submesh via dynamic programming. The quick recognition of small-width submeshes is crucial in such applications as well and the heuristic introduced in this paper would be an indispensable tool.

Although the effectiveness of our heuristic has been verified by some test instances and through an application, there is no theoretical guarantee on the quality of the decomposition it produces. In fact, for any given $r > 0$, it is easy to construct an example for which our heuristic produces a decomposition with width that is more than r times greater than the optimum width. A fast algorithm with some guarantee on the quality of decomposition would be of great interest.

Acknowledgment. This work was done while the author was visiting Max Planck Institute for Computer Science at Saarbrücken. He thanks its algorithm group for the hospitality and for the stimulating environment.

References

1. S. Arnborg and J. Lagergren and D. Seese, Easy problems for tree-decomposable graphs, *Journal of Algorithms*, 12, 308–340, 1991
2. H. Bodlaender, A tourist guide through treewidth, *Acta Cybernetica*, 11, 1–21, 1993.
3. W. Cook and P.D. Seymour, Tour merging via branch-decomposition, to appear in *INFORMS Journal on Computing*, 2003
4. I.V. Hicks, Branch Decompositions and their applications, Phd thesis, Rice University, April, 2000.
5. N. Robertson and P.D. Seymour, Graph minors II. Algorithmic aspects of tree width, *Journal of Algorithms*, 7, 309–322, 1986
6. N. Robertson and P.D. Seymour, Graph minors X. Obstructions to tree-decomposition, *Journal of Combinatorial Theory, Series B*, 153–190, 1991
7. P.D. Seymour and R. Thomas, Call routing and the ratcatcher. *Combinatorica*, 14(2), 217–241, 1994
8. H. Tamaki, A linear time heuristic for the branch-decomposition of planar graphs, Max Planck Institute Research Report, MPI-I-1-03-010, 2003.
9. H. Tamaki, Solving TSP on sparse planar graphs, in preparation.
10. TSP home page by D. Applegate, R. Bixby, W. Cook and V. Chvátal:
<http://www.math.princeton.edu/tsp/>
11. TSPLIB by Gerd Reinelt:
<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>
12. Planar graph test data by I.V.Hicks:
<http://ie.tamu.edu/People/faculty/Hicks/data.html>

Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs^{*}

Dorothea Wagner and Thomas Willhalm

Universität Karlsruhe,

Institut für Logik, Komplexität und Deduktionssysteme, D-76128 Karlsruhe

Abstract. In this paper, we consider Dijkstra's algorithm for the single source single target shortest paths problem in large sparse graphs. The goal is to reduce the response time for online queries by using precomputed information. For the result of the preprocessing, we admit at most linear space. We assume that a layout of the graph is given. From this layout, in the preprocessing, we determine for each edge a geometric object containing all nodes that can be reached on a shortest path starting with that edge. Based on these geometric objects, the search space for online computation can be reduced significantly. We present an extensive experimental study comparing the impact of different types of objects. The test data we use are traffic networks, the typical field of application for this scenario.

1 Introduction

In this paper, we consider the problem to answer a lot of single-source single-target shortest-path queries in a large graph. We admit an expensive preprocessing in order to speed up the query time. From a given layout of the graph we extract geometric information that can be used online to answer the queries. (See also [1,2].) A typical application of this problem is a route planning system for cars, bikes and hikers or scheduled vehicles like trains and busses. Usually, variants of Dijkstra's algorithm are used to realize such systems.

In the comparison model, Dijkstra's algorithm [3] with Fibonacci heaps [4] is still the fastest known algorithm for the general case of arbitrary non-negative edge lengths. Algorithms with worst case linear time are known for undirected graphs and integer weights [5]. Algorithms with average case linear time are known for edge weights uniformly distributed in the interval $[0, 1]$ [6] and for edge weights uniformly distributed in $\{1, \dots, M\}$ [7]. A recent study [8] shows that the sorting bottleneck can be also avoided in practice for undirected graphs.

The application of shortest path computations in travel networks is also widely covered by the literature: [9] compares different algorithms to compute shortest paths trees in real road networks. In [10] the two best label setting

^{*} This work was partially supported by the Human Potential Programme of the European Union under contract no. HPRN-CT-1999-00104 (AMORE) and by the DFG under grant WA 654/12-1.

and label correcting algorithms were compared in respect to single source single target shortest path computations. They conclude that for shorter paths a label correcting algorithm should be preferred. In [11] Barrett et al. present a system that covers formal language constraints and changes over time. Modeling an (interactive) travel information system for scheduled transport is covered by [12] and [13], and a multi-criteria implementation is presented in [14]. A distributed system which integrates multiple transport providers has been realized in [15].

One of the features of travel planning (independent of the vehicle type) is the fact that the network does not change for a certain period of time while there are many queries for shortest paths. This justifies a heavy preprocessing of the network to speed up the queries. Although pre-computing and storing the shortest paths for all pairs of nodes would give us “constant-time” shortest-path queries, the quadratic space requirement for traffic networks with 10^5 and more nodes prevents us from doing so.

In this paper, we explore the possibility to reduce the search space of Dijkstra’s algorithm by using precomputed information that can be stored in $O(n + m)$ space. In fact, this paper shows that storing partial results reduces the number of nodes visited by Dijkstra’s algorithm to 10%. (Figure 1 gives an illustrative example.) We use a very fundamental observation on shortest paths. In general, an edge that is not the first edge on a shortest path to the target can be ignored safely in any shortest path computation to this target. More precisely, we apply the following concept: In the preprocessing, for each edge e , the set of nodes $S(e)$ is stored that can be reached by a shortest path starting with e . While running Dijkstra’s algorithm, edges e for which the target is not in $S(e)$ are ignored.

As storing all sets $S(e)$ would need $O(mn)$ space, we relax the condition by storing a geometric object for each edge that contains *at least* $S(e)$. Remark that this does in fact still lead to a correct result, but may increase the number of visited nodes to more than the strict minimum (i.e. the number of nodes in the shortest path). In order to generate the geometric objects, a layout $L : V \rightarrow \mathbb{R}^2$ is used. For the application of travel information systems, such a layout is for example given by the geographic locations of the nodes. It is however not required that the edge lengths are derived from the layout. In fact, for some of our experimental data this is even not the case.

In [1] angular sectors were introduced for the special case of a time table information system. Our results are more general in two respects: We examine the impact of various different geometric objects and consider Dijkstra for general embedded graphs. It turns out that a significant improvement can be achieved by using other geometric objects than angular sectors. Actually, in some cases the speed-up is even a factor of about two.

The next section contains – after some definitions – a formal description of our shortest path problem. Sect. 3 gives precise arguments how and why the pruning of edges works. In Sect. 4, we describe the geometric objects and the test graphs that we used in our experiments. The statistics and results are presented and discussed in the last section before the summary.

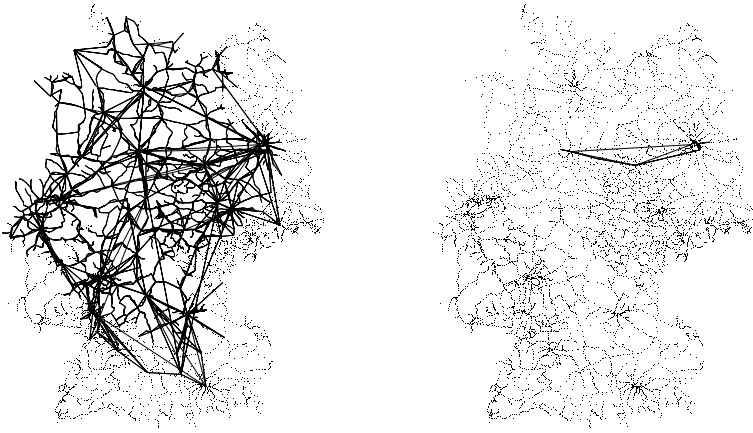


Fig. 1. The search space for a query from Hannover to Berlin for DIJKSTRA’S ALGORITHM (left) and DIJKSTRA’S ALGORITHM WITH PRUNING using bounding boxes (right).

2 Definitions and Problem Description

2.1 Graphs

A directed *graph* G is a pair (V, E) , where V is a finite set and $E \subseteq V \times V$. The elements of V are the *nodes* and the elements of E are the *edges* of the graph G . Throughout this paper, the number of nodes $|V|$ is denoted by n and the number of edges $|E|$ is denoted by m . A *path* in G is a sequence of nodes $u_1, \dots, u_k$ such that $(u_i, u_{i+1}) \in E$ for all $1 \leq i < k$. A path with $u_1 = u_k$ is called a *cycle*. The edges of a graph are *weighted* by a function $w : E \rightarrow \mathbb{R}$. We interpret the weights as edge lengths in the sense that the *length of a path* is the sum of the weights of its edges.

If n denotes the number of nodes, a graph can have up to n^2 edges. We call a graph *sparse*, if the number of edges m is in $o(n^2)$, and we call a graph *large*, if one can only afford a memory consumption that is linear in the size of the graph $O(n + m)$. In particular for large sparse graphs, n^2 space is not affordable.

2.2 Shortest Path Problem

Given a weighted graph $G = (V, E)$, $w : E \rightarrow \mathbb{R}$, the *single source single target shortest paths problem* consists in finding a shortest path from a given source $s \in V$ to a given target $t \in V$. Note that the problem is only well defined for all pairs, if G does not contain negative cycles. Using Johnson’s algorithm [16] it is possible to convert in $O(nm \log n)$ time the edge weights $w : E \rightarrow \mathbb{R}$ to non-negative edge weights $w' : E \rightarrow \mathbb{R}$ that result in the same shortest paths. We will therefore assume in the rest of this paper, that edge weights are non-negative.

A closer look at DIJKSTRA’S ALGORITHM (Algorithm 1) reveals that one needs to visit all nodes of the graph to initialize the marker “visited”, which


```

1 insert source  $s$  in priority queue  $Q$  and set  $\text{dist}(s) := 0$ 
2 while target  $t$  is not marked as finished and priority queue is not empty
3   get node  $u$  with highest priority in  $Q$  and mark it as finished
4   for all neighbor nodes  $v$  of  $u$ 
5     set  $\text{new-dist} := \text{dist}(u) + w((u, v))$ 
6     if neighbor node  $v$  is unvisited
7       set  $\text{dist}(v) := \text{new-dist}$ 
8       insert neighbor node  $v$  in priority queue with priority  $-\text{dist}(v)$ 
9       mark neighbor node  $v$  as visited
10    else
11      if  $\text{dist}(v) > \text{new-dist}$ 
12        set  $\text{dist}(v) := \text{new-dist}$ 
13        increase priority of neighbor node to  $-\text{dist}(v)$ 

```

Algorithm 1: DIJKSTRA'S ALGORITHM

obviously prevents a sub-linear query time. This shortcoming can be eliminated by introducing a global integer variable “time” and replacing the marker “visited” by a time stamp for every node. These time stamps are set to zero in the preprocessing. The time variable is incremented in each run of the algorithm and, instead of marking a node as visited, the time stamp of the node is set to the current time. The test whether a node has been visited in the current run of the algorithm can then be replaced by a comparison of the time stamp with the current time. (For sake of clarity, we didn't include this technique in the pseudo-code.)

3 Geometric Pruning

3.1 Pruning the Search Space

The goal of DIJKSTRA'S ALGORITHM WITH PRUNING, see Algorithm 2, is to decrease the number of visited nodes, the “search space”, by visiting only a subset of the neighbors (line 4a). The idea is illustrated in Fig. 2. The condition is formalized by the notion of a consistent container:

Definition 1. *Given a weighted graph $G = (V, E)$, $w : E \rightarrow \mathbb{R}_0^+$, we call a set of nodes $C \subseteq V$ a container. A container C associated with an edge (u, v) is called consistent, if for all shortest paths from u to t that start with the edge (u, v) , the target t is in C .*

In other words, $C(u, v)$ is consistent, if $S(u, v) \subseteq C(u, v)$.

Theorem 1. *Given a weighted graph $G = (V, E)$, $w : E \rightarrow \mathbb{R}_0^+$ and for each edge e a consistent container $C(e)$, then DIJKSTRA'S ALGORITHM WITH PRUNING finds a shortest path from s to t .*

```

1 insert source  $s$  in priority queue  $Q$  and set  $\text{dist}(s) := 0$ 
2 while target  $t$  is not marked as finished and priority queue is not empty
3   get node  $u$  with highest priority in  $Q$  and mark it as finished
4   for all neighbor nodes  $v$  of  $u$ 
4a     if  $t \in C(u, v)$ 
5       set  $\text{new-dist} := \text{dist}(u) + w((u, v))$ 
6       if neighbor node  $v$  is unvisited
7         set  $\text{dist}(v) := \text{new-dist}$ 
8         insert neighbor node  $v$  in priority queue with priority  $-\text{dist}(v)$ 
9         mark neighbor node  $v$  as visited
10      else
11        if  $\text{dist}(v) > \text{new-dist}$ 
12          set  $\text{dist}(v) := \text{new-dist}$ 
13          increase priority of neighbor node to  $-\text{dist}(v)$ 

```

Algorithm 2: DIJKSTRA'S ALGORITHM WITH PRUNING. Neighbors are only visited, if the edge (u, v) is in the consistent container $C(u, v)$.

Proof. Consider the shortest path P from s to t that is found by DIJKSTRA'S ALGORITHM. If for all edges $e \in P$ the target node t is in $C(e)$, the path P is found by DIJKSTRA'S ALGORITHM WITH PRUNING, because the pruning does not change the order in which the edges are processed. A sub-path of a shortest path is again a shortest path, so for all $(u, v) \in P$, the sub-path of P from u to t is a shortest u - t -path. Then by definition of consistent container, $t \in C(u, v)$.

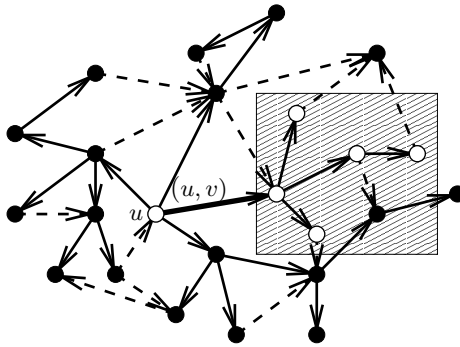


Fig. 2. DIJKSTRA'S ALGORITHM is run for a node $u \in V$. Let the white nodes be those nodes that can be reached on a shortest path using the edge (u, v) . A geometric object is constructed that contains these nodes. It may contain other points, but this does only affect the running time and not the correctness of DIJKSTRA'S ALGORITHM WITH PRUNING.

3.2 Geometric Containers

The containers that we are using are geometric objects. Therefore, we assume that we are given a *layout* $L : V \rightarrow \mathbb{R}^2$ in the Euclidean plane. For ease of notation we will identify a node $v \in V$ with its location $L(v) \in \mathbb{R}^2$ in the plane.

In the previous section, we explained how consistent containers are used to prune the search space of DIJKSTRA'S ALGORITHM. In particular, the correctness of the result does not depend on the layout of the graph that is used to construct the containers. However, the impact of the container for speeding up Dijkstra's Algorithm does depend on the relation of the layout and the edge weights. This section describes the geometric containers that we used in our tests. We require that a container has a description of constant size and that its containment test takes constant time.

Recall that $S(u, v)$ is the set of all nodes t with the property that there is a shortest u - t -path that starts with the edge (u, v) . For some types of containers we need to know $S(u, v)$ explicitly in order to determine the container $C(u, v)$ associated with the edge (u, v) . Therefore we now detail how to compute $S(u, v)$. To determine the sets $S(u, v)$ for every edge $(u, v) \in E$, Dijkstra's algorithm is run for each node u . For each node $t \in V$, we store the edge (u, v) with $t \in S(u, v)$, i.e. the first edge of the u - t -path in the shortest paths tree. This can be done in a similar way one constructs the shortest path tree: In the beginning, all nodes v adjacent to u are associated with the edge (u, v) in an array $A[v]$. Every time the distance label of a node t is adjusted via (p, t) , we update $A[t]$ with $A[p]$, i.e. we associate with t the edge of its predecessor p . When a node is marked as *finished*, $A[t]$ holds the outgoing edge of u with which a shortest path from u to t starts. We can then construct the sets $S(u, v)$ for all edges incident to u . The storage requirement is linear in the number of nodes and the geometric containers can then be easily constructed and attached to the edges.

On the other hand, it is possible for some geometric containers to be constructed without actually creating the sets $S(u, v)$ in memory. These containers can be constructed *online* by adding one point after the other without storing all points explicitly. In other words, there exists an efficient method to update a container $C(u, v)$ with a new point t that has turned out to lie in $S(u, v)$.

Disk Centered at Tail. For each edge (u, v) , the disk with center at u and minimum radius that covers $S(u, v)$ is computed. This is the same as finding the maximal distance of all nodes in $S(u, v)$ from u . The size of such an object is constant, because the only value that needs to be stored is the radius¹ which leads to a space consumption linear in the number of edges. The radius can be determined online by simply increasing it if necessary.

Ellipse. An extension of the disk is the ellipse with foci u and v and minimum radius needed to cover $S(u, v)$. It suffices to remember the radius, which can be found online similarly as in the disk case.

¹ Of course in practice, the squared radius is stored to avoid the computationally expensive square root function.

Angular Sector. Angular sectors are the objects that were used in [1]. For each edge (u, v) a node p left of (u, v) and a node q right of (u, v) are determined such that all nodes in $S(u, v)$ lie within the angular sector $\angle(p, u, q)$. The nodes p and q are chosen in a way that minimizes the angle $\angle(p, u, q)$. They can be determined in an online fashion: If a new point w is outside the angular sector $\angle(p, u, q)$, we set $p := w$ if w is left of (u, v) and $q := w$ if w is right of it. (Note that this is not necessarily the minimum angle at u that contains all points in $S(u, v)$.)

Circular Sector. Angular sectors have the big disadvantage that they are not bounded. By intersecting an angular sector with a disk at the tail of the edge, we get a circular sector. Obviously the minimal circular sector can be found online and needs only constant space (two points and the radius).

Smallest Enclosing Disk. The smallest enclosing disk is the unique disk with smallest area that includes all points. We use the implementation in CGAL [17] of Welzl's algorithm [18] with expected linear running time. The algorithm works offline and storage requirement is at most three points.

Smallest Enclosing Ellipse. The smallest enclosing ellipse is a generalization of the smallest enclosing disk. Therefore, the search space using this container will be at most as large as for smallest enclosing disks (although the actual running time might be larger since the inclusion test is more expensive). Again, Welzl's algorithm is used. The space requirement is constant.

Bounding Box (Axis-Parallel Rectangle). This is the simplest object in our collection. It suffices to store four numbers for each object, which are the lower, upper, left and right boundary of the box. The bounding boxes can easily be computed online while the shortest paths are computed in the preprocessing.

Edge-Parallel Rectangle. Such a rectangle is not parallel to an axis, but to the edge to which it belongs. So, for each edge, the coordinate system is rotated and then the bounding box is determined in this rotated coordinate system. Our motivation to implement this container was the insight that the target nodes for an edge are usually situated in the direction of the edge. A rectangle that targets in this direction might therefore be a better model for the geometric region than one that is parallel to the axes. Note that storage requirements are actually the same as for a bounding box, but additional computations are required to rotate the coordinate system.

Intersection of Rectangles. The rectangle parallel to the axes and the rectangle parallel to the edge are intersected, which should lead to a smaller object. The space consumption of the two objects sum up, but are still constant, and as both objects can be computed online the intersection can be as well.

Smallest Enclosing Rectangle. If we allow a rectangle to be oriented in any direction and search for one with smallest area, things are not as simple anymore. The algorithm from [19] finds it in linear time. However, due to numerical inconsistencies we had to incorporate additional tests to assure that all points are in fact inside the rectangle. As for the minimal enclosing

disk, this container has to be calculated offline, but needs only constant space for its orientation and dimensions.

Smallest Enclosing Parallelogram. Toussaint's idea to use rotating calipers has been extended in [20] to find the smallest enclosing parallelogram. Space consumption is constant and the algorithm is offline.

Convex Hull. The convex hull does not fulfill our requirement that containers must be of constant size. It is included here, because it provides a lower bound for all convex objects. If there is a best convex container, it cannot exclude more points than the convex hull.

4 Implementation and Experiments

We implemented the algorithm in C++ using the GNU compiler g++ 2.95.3. We used the graph data structure from LEDA 4.3 (see [21]) as well as the Fibonacci heaps and the convex hull algorithm provided. I/O was done by the LEDA extension package for GraphML with Xerces 2.1. For the minimal disks, ellipses and parallelograms, we used CGAL 2.4. In order to perform efficient containment tests for minimal disks, we converted the result from arbitrary precision to built-in doubles. To overcome numerical inaccuracies, the radius was increased if necessary to guarantee that all points are in fact inside the container. For minimal ellipses, we used arbitrary precision which affects the running time but not the search space. Instead of calculating the minimal disk (or ellipse) of a point set, we determine the minimal disk (or ellipse) of the convex hull. This speeds up the preprocessing for these containers considerably. Although CGAL also provides an algorithm for minimal rectangles, we decided to implement one ourselves, because one cannot simply increase a radius in this case. Due to numeric instabilities, our implementation does not guarantee to find the minimal container, but asserts that all points are inside the container. We computed the convex hulls with LEDA [21]. The experiments were performed on an Intel Xeon with 2.4 GHz on the Linux 2.4 platform.

It is crucial to this problem to do the statistics with data that stem from real applications. We are using two types of data:

Street Networks. We have gathered street maps from various public Internet servers. They cover some American cities and their surroundings. Unfortunately the maps did not contain more information than the mere location of the streets. In particular, streets are not distinguished from freeways, and one-way streets are not marked as such, which makes these graphs bidirected with the Euclidean edge length. The street networks are typically very sparse with an average degree hardly above 2. The size of these networks varies from 1444 to 20466 nodes.

Railway Networks. The railway networks of different European countries were derived from the winter 1996/1997 time table. The nodes of such a graph are the stations and an edge between two stations exists iff there is a non-stop connection. The edges are weighted by the average travel time. In particular, here the weights do *not* directly correspond to the layout. They

have between 409 nodes (Netherlands) and 6884 nodes (Germany) but are not as sparse as the street graphs.

All test sets were converted to the XML-based GraphML file format [22] to allow us a unified processing.

We sampled random single source single target queries to determine the average number of nodes that are visited by the algorithm. The sampling was done until the length of the 95% confidence interval was smaller than 5% of the average search space. The length of the confidence interval is $2t_{n-1, 1-\frac{\alpha}{2}} sn^{-\frac{1}{2}}$, where $t_{n-1, 1-\frac{\alpha}{2}}$ denotes the upper critical error for the t -distribution with $n-1$ degrees of freedom. For $n > 100$ we approximated the t -distribution by the normal distribution. Note that the sample mean $\bar{x}$ and the standard error s can be calculated recursively:

$$\bar{x}_{(k)} = \frac{1}{k} (\bar{x}_{(k-1)}(k-1) + x_k)$$

$$s_{(k)}^2 = \frac{1}{k-1} \left[(k-2)s_{(k-1)}^2 + (k-1)\bar{x}_{(k-1)}^2 + x_k^2 - k\bar{x}_{(k)}^2 \right]$$

where the subscript (k) marks the mean value and standard error for k samples, respectively. Using these formulas, it is possible to run random single-source single-target shortest-path queries until the confidence interval is small enough.

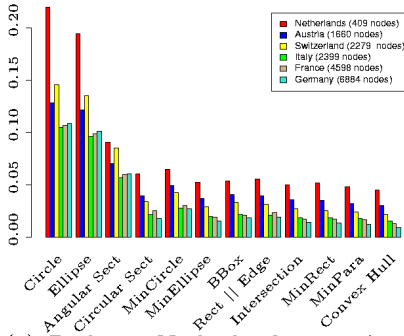
5 Results and Discussion

Figure 3(a) depicts the results for railway networks. The average number of nodes that the algorithm visited are shown. To enable the comparison of the result for different graphs, the numbers are relative to the average search space of DIJKSTRA'S ALGORITHM (without pruning). As expected, the pruning for disks around the tail is by far not as good as the other methods. As expected, the pruning for disks around the tail is by far not as good as the other methods. Note however, that the average search space is still reduced to about 10%. The only type of objects studied previously [1], the angular sectors, result in a reduction to about 6%, but, if both are intersected, we get only 3.5%. Surprisingly the result for bounding boxes is about the same as for the better tailored circular sectors, directed and even minimal rectangles or parallelograms.

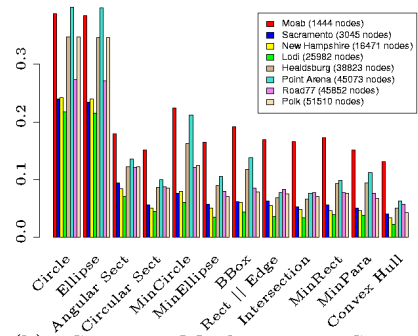
Of course the results of more general containers like minimal rectangle vs. bounding box are better, but the differences are not very big. Furthermore, the difference to our lower bound for convex objects is comparatively small.

The data sets are ordered according to their size. In most cases the speed-up is better the larger the graph. This can be explained by the observation that the search space of a lot of queries is already limited by the size of the graph.

Apart from the number of visited nodes, we examined the average running time. We depict them in Fig. 4, again relative to the running time of the unmodified Dijkstra. It is obvious that the slightly smaller search space for the more complicated containers does not pay off. In fact the simplest container, the axis-parallel bounding box, results in the fastest algorithm. For this container a preprocessing is useful for Germany, if more than 18000 queries are performed.

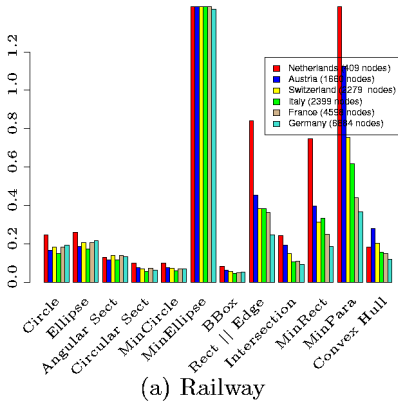


(a) Railway: Netherlands 409, Austria 1660, Switzerland 2279, Italy 2399, France 4598, Germany 6884

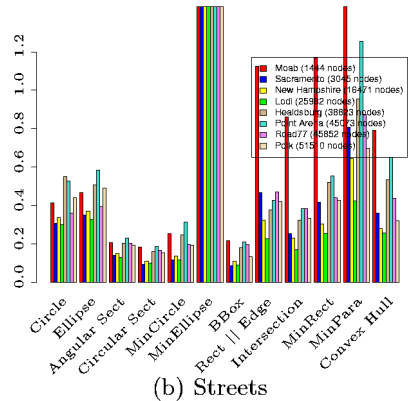


(b) Streets: Moab 1444, Sacramento 3045, New Hampshire 16471, Lodi 25982, Healdsburg 38823, Point Arena 45073, Road77 45852, Polk 51510

Fig. 3. Average number of visited nodes relative to DIJKSTRA's ALGORITHM for all graphs and geometric objects. The graphs are ordered according to the number of nodes.



(a) Railway



(b) Streets

Fig. 4. Average query running time relative to DIJKSTRA's ALGORITHM for all data sets and geometric objects. The values for minimal ellipse and minimal parallelogram are clipped. They use arbitrary precision and are therefore much slower than the other containment tests.

6 Conclusion

We have seen that using a layout may lead to a considerably speed-up, if one allows a preprocessing. Actually, we are able to reduce the search space to 5–10% while even “bad” containers result in a reduction to less than 50%. The somewhat

surprising result is that the simple bounding box outperforms other geometric objects in terms of CPU cycles in a lot of cases.

The presented technique can easily be combined with other methods:

- The geometric pruning is in fact independent of the priority queue. Algorithms using a special priority queue such as [6,7] can easily be combined with it. The decrease of the search space is in fact the same (but the actual running time would be different of course).
- Goal-directed search [23] or A^* has been shown in [24,25] to be very useful for transportation networks. As it simply modifies the edge weights, a combination of geometric pruning and A^* can be realized straight forward.
- Bidirectional search [26] can be integrated by reverting all edges and running the preprocessing a second time. Space and time consumption for the preprocessing simply doubles.
- In combination with a multi-level approach [27,2], one constructs a graph containing all levels and inter-level edges first. The geometric pruning is then performed on this graph.

Acknowledgment. We thank Jasper Möller for his help in implementing and Alexander Wolff for many helpful comments regarding the presentation of this paper.

References

1. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *Journal of Experimental Algorithmics* **5** (2000)
2. Schulz, F., Wagner, D., Zaroliagis, C.: Using multi-level graphs for timetable information. In: *Proc. Algorithm Engineering and Experiments (ALENEX '02)*. LNCS, Springer (2002) 43–59
3. Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numerische Mathematik* **1** (1959) 269–271
4. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)* **34** (1987) 596–615
5. Thorup, M.: Undirected single source shortest path in linear time. In: *IEEE Symposium on Foundations of Computer Science*. (1997) 12–21
6. Meyer, U.: Single-source shortest-paths on arbitrary directed graphs in linear average-case time. In: *Symposium on Discrete Algorithms*. (2001) 797–806
7. Goldberg, A.V.: A simple shortest path algorithm with linear average time. In: *auf der Heide, F.M., ed.: ESA 2001*. Volume 2161 of LNCS., Springer (2001) 230–241
8. Pettie, S., Ramachandran, V., Sridhar, S.: Experimental evaluation of a new shortest path algorithm. In: *ALENEX'02*. (2002) 126–142
9. Zahn, F.B., Noon, C.E.: Shortest path algorithms: An evaluation using real road networks. *Transportation Science* **32** (1998) 65–73
10. Zahn, F.B., Noon, C.E.: A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths. *Journal of Geographic Information and Decision Analysis* **4** (2000)

11. Barrett, C., Bisset, K., Jacob, R., Konjevod, G., Marathe, M.: Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the transims router. In Möhring, R., Raman, R., eds.: *ESA 2002*. Volume 2461 of LNCS., Springer (2002) 126–138
12. Siklóssy, L., Tulp, E.: Trains, an active time-table searcher. In: *Proc. 8th European Conf. Artificial Intelligence*. (1988) 170–175
13. Nachtigall, K.: Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research* **83** (1995) 154–166
14. Müller-Hannemann, M., Weihe, K.: Pareto shortest paths is often feasible in practice. In Brodal, G., Frigioni, D., Marchetti-Spaccamela, A., eds.: *WAE 2001*. Volume 2461 of LNCS., Springer (2001) 185–197
15. Preuss, T., Syrbe, J.H.: An integrated traffic information system. In: *Proc. 6th Int. Conf. Appl. Computer Networking in Architecture, Construction, Design, Civil Eng., and Urban Planning (europIA '97)*. (1997)
16. Johnson, D.B.: Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)* **24** (1977) 1–13
17. Fabri, A., Giezeman, G.J., Kettner, L., Schirra, S., Schönherr, S.: On the design of CGAL a computational geometry algorithms library. *Softw. – Pract. Exp.* **30** (2000) 1167–1202
18. Welzl, E.: Smallest enclosing disks (balls and ellipsoids). In Maurer, H., ed.: *New Results and New Trends in Computer Science*. LNCS. Springer (1991)
19. Toussaint, G.: Solving geometric problems with the rotating calipers. In Protonotarios, E.N., ed.: *MELECON 1983*, NY, IEEE (1983) A10.02/1–4
20. Schwarz, C., Teich, J., Vainshtein, A., Welzl, E., Evans, B.L.: Minimal enclosing parallelogram with application. In: *Proceedings of the eleventh annual symposium on Computational geometry*, ACM Press (1995) 434–435
21. Mehlhorn, K., Näher, S.: *LEDA, A platform for Combinatorial and Geometric Computing*. Cambridge University Press (1999)
22. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Scott, M.: GraphML progress report. In Mutzel, P., Jünger, M., Leipert, S., eds.: *GD 2001*. Volume 2265 of LNCS., Springer (2001) 501–512
23. Sedgewick, R., Vitter, J.S.: Shortest paths in euclidean space. *Algorithmica* **1** (1986) 31–48
24. Shekhar, S., Kohli, A., Coyle, M.: Path computation algorithms for advanced traveler information system (atis). In: *Proc. 9th IEEE Intl. Conf. Data Eng.* (1993) 31–39
25. Jacob, R., Marathe, M., Nagel, K.: A computational study of routing algorithms for realistic transportation networks. In Mehlhorn, K., ed.: *WAE'98*. (1998)
26. Pohl, I.: Bi-directional search. In Meltzer, B., Michie, D., eds.: *Sixth Annual Machine Intelligence Workshop*. Volume 6 of *Machine Intelligence.*, Edinburgh University Press (1971) 137–140
27. Jung, S., Pramanik, S.: Hiti graph model of topographical road maps in navigation systems. In: *Proc. 12th IEEE Int. Conf. Data Eng.* (1996) 76–84

Table of Contents

Invited Lectures

Sublinear Computing	1
<i>Bernard Chazelle</i>	
Authenticated Data Structures	2
<i>Roberto Tamassia</i>	
Approximation Algorithms and Network Games	6
<i>Éva Tardos</i>	

Contributed Papers: Design and Analysis Track

I/O-Efficient Structures for Orthogonal Range-Max and Stabbing-Max Queries	7
<i>Pankaj K. Agarwal, Lars Arge, Jun Yang, Ke Yi</i>	
Line System Design and a Generalized Coloring Problem	19
<i>Mansoor Alicherry, Randeep Bhatia</i>	
Lagrangian Relaxation for the k-Median Problem: New Insights and Continuity Properties	31
<i>Aaron Archer, Ranjithkumar Rajagopalan, David B. Shmoys</i>	
Scheduling for Flow-Time with Admission Control	43
<i>Nikhil Bansal, Avrim Blum, Shuchi Chawla, Kedar Dhamdhere</i>	
On Approximating a Geometric Prize-Collecting Traveling Salesman Problem with Time Windows	55
<i>Reuven Bar-Yehuda, Guy Even, Shimon (Moni) Shahar</i>	
Semi-clairvoyant Scheduling	67
<i>Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Kirk Pruhs</i>	
Algorithms for Graph Rigidity and Scene Analysis	78
<i>Alex R. Berg, Tibor Jordán</i>	
Optimal Dynamic Video-on-Demand Using Adaptive Broadcasting	90
<i>Therese Biedl, Erik D. Demaine, Alexander Golynski, Joseph D. Horton, Alejandro López-Ortiz, Guillaume Poirier, Claude-Guy Quimper</i>	

Multi-player and Multi-round Auctions with Severely Bounded Communication	102
<i>Liad Blumrosen, Noam Nisan, Ilya Segal</i>	
Network Lifetime and Power Assignment in ad hoc Wireless Networks ...	114
<i>Gruia Calinescu, Sanjiv Kapoor, Alexander Olshevsky, Alexander Zelikovsky</i>	
Disjoint Unit Spheres Admit at Most Two Line Transversals	127
<i>Otfried Cheong, Xavier Goaoc, Hyeon-Suk Na</i>	
An Optimal Algorithm for the Maximum-Density Segment Problem	136
<i>Kai-min Chung, Hsueh-I Lu</i>	
Estimating Dominance Norms of Multiple Data Streams	148
<i>Graham Cormode, S. Muthukrishnan</i>	
Smoothed Motion Complexity	161
<i>Valentina Damerow, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, Christian Sohler</i>	
Kinetic Dictionaries: How to Shoot a Moving Target	172
<i>Mark de Berg</i>	
Deterministic Rendezvous in Graphs	184
<i>Anders Dessmark, Pierre Fraigniaud, Andrzej Pelc</i>	
Fast Integer Programming in Fixed Dimension	196
<i>Friedrich Eisenbrand</i>	
Correlation Clustering – Minimizing Disagreements on Arbitrary Weighted Graphs	208
<i>Dotan Emanuel, Amos Fiat</i>	
Dominating Sets and Local Treewidth	221
<i>Fedor V. Fomin, Dimitrios M. Thilikos</i>	
Approximating Energy Efficient Paths in Wireless Multi-hop Networks	230
<i>Stefan Funke, Domagoj Matijevic, Peter Sanders</i>	
Bandwidth Maximization in Multicasting	242
<i>Naveen Garg, Rohit Khandekar, Keshav Kunal, Vinayaka Pandit</i>	
Optimal Distance Labeling for Interval and Circular-Arc Graphs	254
<i>Cyril Gavoille, Christophe Paul</i>	
Improved Approximation of the Stable Marriage Problem	266
<i>Magnús M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, Hiroki Yanagisawa</i>	

Fast Algorithms for Computing the Smallest k -Enclosing Disc	278
<i>Sariel Har-Peled, Soham Mazumdar</i>	
The Minimum Generalized Vertex Cover Problem	289
<i>Refael Hassin, Asaf Levin</i>	
An Approximation Algorithm for MAX-2-SAT with Cardinality Constraint	301
<i>Thomas Hofmeister</i>	
On-Demand Broadcasting Under Deadline	313
<i>Bala Kalyanasundaram, Mahe Velauthapillai</i>	
Improved Bounds for Finger Search on a RAM	325
<i>Alexis Kaporis, Christos Makris, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas, Christos Zaroliagis</i>	
The Voronoi Diagram of Planar Convex Objects	337
<i>Menelaos I. Karavelas, Mariette Yvinec</i>	
Buffer Overflows of Merging Streams	349
<i>Alex Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir</i>	
Improved Competitive Guarantees for QoS Buffering	361
<i>Alex Kesselman, Yishay Mansour, Rob van Stee</i>	
On Generalized Gossiping and Broadcasting	373
<i>Samir Khuller, Yoo-Ah Kim, Yung-Chun (Justin) Wan</i>	
Approximating the Achromatic Number Problem on Bipartite Graphs . . .	385
<i>Guy Kortsarz, Sunil Shende</i>	
Adversary Immune Leader Election in ad hoc Radio Networks	397
<i>Miroslaw Kutylowski, Wojciech Rutkowski</i>	
Universal Facility Location	409
<i>Mohammad Mahdian, Martin Pál</i>	
A Method for Creating Near-Optimal Instances of a Certified Write-All Algorithm	422
<i>Grzegorz Malewicz</i>	
I/O-Efficient Undirected Shortest Paths	434
<i>Ulrich Meyer, Norbert Zeh</i>	
On the Complexity of Approximating TSP with Neighborhoods and Related Problems	446
<i>Shmuel Safra, Oded Schwartz</i>	

A Lower Bound for Cake Cutting	459
<i>Jiří Sgall, Gerhard J. Woeginger</i>	
Ray Shooting and Stone Throwing	470
<i>Micha Sharir, Hayim Shaul</i>	
Parameterized Tractability of Edge-Disjoint Paths on Directed Acyclic Graphs	482
<i>Aleksandrs Slivkins</i>	
Binary Space Partition for Orthogonal Fat Rectangles	494
<i>Csaba D. Tóth</i>	
Sequencing by Hybridization in Few Rounds.....	506
<i>Dekel Tsur</i>	
Efficient Algorithms for the Ring Loading Problem with Demand Splitting	517
<i>Biing-Feng Wang, Yong-Hsian Hsieh, Li-Pu Yeh</i>	
Seventeen Lines and One-Hundred-and-One Points	527
<i>Gerhard J. Woeginger</i>	
Jacobi Curves: Computing the Exact Topology of Arrangements of Non-singular Algebraic Curves	532
<i>Nicola Wolpert</i>	

Contributed Papers: Engineering and Application Track

Streaming Geometric Optimization Using Graphics Hardware	544
<i>Pankaj K. Agarwal, Shankar Krishnan, Nabil H. Mustafa, Suresh Venkatasubramanian</i>	
An Efficient Implementation of a Quasi-polynomial Algorithm for Generating Hypergraph Transversals	556
<i>E. Boros, K. Elbassioni, V. Gurvich, Leonid Khachiyan</i>	
Experiments on Graph Clustering Algorithms	568
<i>Ulrik Brandes, Marco Gaertler, Dorothea Wagner</i>	
More Reliable Protein NMR Peak Assignment via Improved 2-Interval Scheduling	580
<i>Zhi-Zhong Chen, Tao Jiang, Guohui Lin, Romeo Rizzi, Jianjun Wen, Dong Xu, Ying Xu</i>	
The Minimum Shift Design Problem: Theory and Practice	593
<i>Luca Di Gaspero, Johannes Gärtner, Guy Kortsarz, Nysret Musliu, Andrea Schaerf, Wolfgang Slany</i>	

Loglog Counting of Large Cardinalities	605
<i>Marianne Durand, Philippe Flajolet</i>	
Packing a Trunk	618
<i>Friedrich Eisenbrand, Stefan Funke, Joachim Reichel, Elmar Schömer</i>	
Fast Smallest-Enclosing-Ball Computation in High Dimensions	630
<i>Kaspar Fischer, Bernd Gärtner, Martin Kutz</i>	
Automated Generation of Search Tree Algorithms for Graph Modification Problems	642
<i>Jens Gramm, Jiong Guo, Falk Hüffner, Rolf Niedermeier</i>	
Boolean Operations on 3D Selective Nef Complexes: Data Structure, Algorithms, and Implementation	654
<i>Miguel Granados, Peter Hachenberger, Susan Hert, Lutz Kettner, Kurt Mehlhorn, Michael Seel</i>	
Fleet Assignment with Connection Dependent Ground Times	667
<i>Sven Grothklags</i>	
A Practical Minimum Spanning Tree Algorithm Using the Cycle Property	679
<i>Irit Katriel, Peter Sanders, Jesper Larsson Träff</i>	
The Fractional Prize-Collecting Steiner Tree Problem on Trees	691
<i>Gunnar W. Klau, Ivana Ljubić, Petra Mutzel, Ulrich Pferschy, René Weiskircher</i>	
Algorithms and Experiments for the Webgraph	703
<i>Luigi Laura, Stefano Leonardi, Stefano Millozzi, Ulrich Meyer, Jop F. Sibeyn</i>	
Finding Short Integral Cycle Bases for Cyclic Timetabling	715
<i>Christian Liebchen</i>	
Slack Optimization of Timing-Critical Nets	727
<i>Matthias Müller-Hannemann, Ute Zimmermann</i>	
Multisampling: A New Approach to Uniform Sampling and Approximate Counting	740
<i>Piotr Sankowski</i>	
Multicommodity Flow Approximation Used for Exact Graph Partitioning	752
<i>Meinolf Sellmann, Norbert Sensen, Larissa Timajev</i>	

A Linear Time Heuristic for the Branch-Decomposition of Planar
Graphs 765
 Hisao Tamaki

Geometric Speed-Up Techniques for Finding Shortest Paths in Large
Sparse Graphs 776
 Dorothea Wagner, Thomas Willhalm

Author Index 789

Author Index

- Agarwal, Pankaj K. 7, 544
Alicherry, Mansoor 19
Archer, Aaron 31
Arge, Lars 7

Bansal, Nikhil 43
Bar-Yehuda, Reuven 55
Becchetti, Luca 67
Berg, Alex R. 78
Berg, Mark de 172
Bhatia, Randeep 19
Biedl, Therese 90
Blum, Avrim 43
Blumrosen, Liad 102
Boros, E. 556
Brandes, Ulrik 568

Calinescu, Gruiua 114
Chawla, Shuchi 43
Chazelle, Bernard 1
Chen, Zhi-Zhong 580
Cheong, Otfried 127
Chung, Kai-min 136
Cormode, Graham 148

Damerow, Valentina 161
Demaine, Erik D. 90
Dessmark, Anders 184
Dhamdhere, Kedar 43
Durand, Marianne 605

Eisenbrand, Friedrich 196, 618
Elbassioni, K. 556
Emanuel, Dotan 208
Even, Guy 55

Fiat, Amos 208
Fischer, Kaspar 630
Flajolet, Philippe 605
Fomin, Fedor V. 221
Fraigniaud, Pierre 184
Funke, Stefan 230, 618

Gaertler, Marco 568
Gärtner, Bernd 630
Gärtner, Johannes 593
Garg, Naveen 242
Gaspero, Luca Di 593

Gavoille, Cyril 254
Goaoc, Xavier 127
Golynski, Alexander 90
Gramm, Jens 642
Granados, Miguel 654
Grothklags, Sven 667
Guo, Jiong 642
Gurvich, V. 556

Hachenberger, Peter 654
Halldórsson, Magnús M. 266
Har-Peled, Sariel 278
Hassin, Refael 289
Hert, Susan 654
Hofmeister, Thomas 301
Horton, Joseph D. 90
Hsieh, Yong-Hsian 517
Hüffner, Falk 642

Iwama, Kazuo 266

Jiang, Tao 580
Jordán, Tibor 78

Kalyanasundaram, Bala 313
Kapoor, Sanjiv 114
Kaporis, Alexis 325
Karavelas, Menelaos I. 337
Katriel, Irit 679
Kesselman, Alex 349, 361
Kettner, Lutz 654
Khachiyan, Leonid 556
Khandekar, Rohit 242
Khuller, Samir 373
Kim, Yoo-Ah 373
Klau, Gunnar W. 691
Kortsarz, Guy 385, 593
Krishnan, Shankar 544
Kunal, Keshav 242
Kutyłowski, Mirosław 397
Kutz, Martin 630

Laura, Luigi 703
Leonardi, Stefano 703
Levin, Asaf 289
Liebchen, Christian 715
Lin, Guohui 580

- Ljubić, Ivana 691
 López-Ortiz, Alejandro 90
 Lotker, Zvi 349
 Lu, Hsueh-I 136

 Mahdian, Mohammad 409
 Makris, Christos 325
 Malewicz, Grzegorz 422
 Mansour, Yishay 349, 361
 Marchetti-Spaccamela, Alberto 67
 Matijevic, Domagoj 230
 Mazumdar, Soham 278
 Mehlhorn, Kurt 654
 Meyer, Ulrich 434, 703
 Meyer auf der Heide, Friedhelm 161
 Millozzi, Stefano 703
 Miyazaki, Shuichi 266
 Müller-Hannemann, Matthias 727
 Musliu, Nysret 593
 Mustafa, Nabil H. 544
 Muthukrishnan, S. 148
 Mutzel, Petra 691

 Na, Hyeon-Suk 127
 Niedermeier, Rolf 642
 Nisan, Noam 102

 Olshevsky, Alexander 114

 Pál, Martin 409
 Pandit, Vinayaka 242
 Patt-Shamir, Boaz 349
 Paul, Christophe 254
 Pelc, Andrzej 184
 Pferschy, Ulrich 691
 Poirier, Guillaume 90
 Pruhs, Kirk 67

 Quimper, Claude-Guy 90

 Räcke, Harald 161
 Rajagopalan, Ranjithkumar 31
 Reichel, Joachim 618
 Rizzi, Romeo 580
 Rutkowski, Wojciech 397

 Safra, Shmuel 446
 Sanders, Peter 230, 679
 Sankowski, Piotr 740
 Schaerf, Andrea 593
 Scheideler, Christian 161
 Schömer, Elmar 618
 Schwartz, Oded 446

 Seel, Michael 654
 Segal, Ilya 102
 Sellmann, Meinolf 752
 Sensen, Norbert 752
 Sgall, Jiří 459
 Shahr, Shimon (Moni) 55
 Sharir, Micha 470
 Shaul, Hayim 470
 Shende, Sunil 385
 Shmoys, David B. 31
 Sibeyn, Jop F. 703
 Sioutas, Spyros 325
 Slany, Wolfgang 593
 Slivkins, Aleksandrs 482
 Sohler, Christian 161
 Stee, Rob van 361

 Tamaki, Hisao 765
 Tamassia, Roberto 2
 Tardos, Éva 6
 Thilikos, Dimitrios M. 221
 Timajev, Larissa 752
 Tóth, Csaba D. 494
 Träff, Jesper Larsson 679
 Tsakalidis, Athanasios 325
 Tschilas, Kostas 325
 Tsur, Dekel 506

 Velauthapillai, Mahe 313
 Venkatasubramanian, Suresh 544

 Wagner, Dorothea 568, 776
 Wan, Yung-Chun (Justin) 373
 Wang, Bing-Feng 517
 Weiskircher, René 691
 Wen, Jianjun 580
 Willhalm, Thomas 776
 Woeginger, Gerhard J. 459, 527
 Wolpert, Nicola 532

 Xu, Dong 580
 Xu, Ying 580

 Yanagisawa, Hiroki 266
 Yang, Jun 7
 Yeh, Li-Pu 517
 Yi, Ke 7
 Yvinec, Mariette 337

 Zaroliagis, Christos 325
 Zeh, Norbert 434
 Zelikovsky, Alexander 114
 Zimmermann, Ute 727