

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Hung Q. Ngo (Ed.)

Computing and Combinatorics

15th Annual International Conference, COCOON 2009
Niagara Falls, NY, USA, July 13-15, 2009
Proceedings



Springer

Volume Editor

Hung Q. Ngo
State University of New York at Buffalo
Department of Computer Science and Engineering
201 Bell Hall, Amherst, NY 14260, USA
E-mail: hungngo@buffalo.edu

Library of Congress Control Number: 2009929548

CR Subject Classification (1998): F.2, G.2, I.3.5, F.1, F.4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-642-02881-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-02881-6 Springer 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. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2009
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12715787 06/3180 5 4 3 2 1 0

Preface

The papers in this volume were selected for presentation at the 15th Annual International Computing and Combinatorics Conference (COCOON 2009), held during July 13–15, 2009 in Niagara Falls, New York, USA. Previous meetings of this conference were held in Xian (1995), Hong Kong (1996), Shanghai (1997), Taipei (1998), Tokyo (1999), Sydney (2000), Guilin (2001), Singapore (2002), Big Sky (2003), Jeju Island (2004), Kunming (2005), Taipei (2006), Alberta (2007), and Dalian (2008).

In response to the Call for Papers, 125 extended abstracts (not counting withdrawn papers) were submitted from 28 countries and regions, of which 51 were accepted. Authors of the submitted papers were from Cyprus (1), The Netherlands (1), Bulgaria (1), Israel (1), Vietnam (2), Finland (1), Puerto Rico (2), Australia (4), Norway (4), Portugal (1), Spain (2), France (16), Republic of Korea (3), Singapore (2), Italy (6), Iran, (4), Greece (7), Poland (4), Switzerland (8), Hong Kong (10), UK (12), India (7), Taiwan (18), Canada (23), China (19), Japan (39), Germany (44), and the USA (77).

The submitted papers were evaluated by an international Technical Program Committee (TPC) consisting of Srinivas Aluru (Iowa State University, USA), Lars Arge (University of Aarhus, Denmark), Vikraman Arvind (Institute of Mathematical Sciences, India), James Aspnes (Yale University, USA), Mikhail Atallah (Purdue University, USA), Gill Barequet (Technion - Israel Institute of Technology, Israel), Michael Brudno (University of Toronto, Canada), Jianer Chen (Texas A&M, USA), Bhaskar DasGupta (University of Illinois at Chicago, USA), Anupam Gupta (Carnegie Mellon University, USA), Lane A. Hemaspaandra (University of Rochester, USA), Kazuo Iwama (Kyoto University, Japan), Avner Magen (University of Toronto, Canada), Peter Bro Miltersen (University of Aarhus, Denmark), Hung Q. Ngo (SUNY Buffalo, USA), Mohammad Salavatipour (University of Alberta, Canada), Alan Selman (SUNY Buffalo, USA), Maria Serna (Universitat Politècnica de Catalunya, Spain), Hans Ulrich Simon (Ruhr University Bochum, Germany), Daniel Stefankovic (University of Rochester, USA), Chaitanya Swamy (University of Waterloo, Canada), My T. Thai (University of Florida, USA), and Philipp Woelfel (University of Calgary, Canada).

Each paper was evaluated by at least three TPC members, with possible assistance of the external referees, as indicated by the referee list found in these proceedings. In addition to the selected papers, the conference also included three invited presentations by Venkat Guruswami (Washington), Muthu Muthukrishnan (Google Research), and Éva Tardos (Cornell). Muthukrishnan also provided an accompanying article included in these proceedings.

I am extremely thankful to all the TPC members, each of whom reviewed about 17 papers despite a very tight schedule. The time and effort spent per

TPC member were tremendous. By the same token, I profusely thank all the external referees who helped review the submissions. The TPC and external referees not only helped select a strong program for the conference, but also gave very informative feedback to the authors of all submitted papers. Many thanks are due to the three invited speakers and all the authors who submitted papers for consideration, all of whom contributed to the quality of COCOON 2009. I would like to extend my deep gratitude to my colleagues Atri Rudra and Sheng Zhong of SUNY Buffalo, who did a great job in logistically arranging the meeting.

I am deeply grateful for the financial and moral support from the department of Computer Science and Engineering of the State University of New York at Buffalo.

Last but not least, thank you all COCOON 2009 attendees and authors. It was you who made the meeting a success. I sincerely hope that you enjoyed the technical presentations and discussions with your colleagues, and that you will continue to support future COCOONs.

July 2009

Hung Q. Ngo

Organization

COCOON 2009 was sponsored by the department of Computer Science and Engineering of the State University of New York at Buffalo.

Executive Committee

Conference and Program Chair	Hung Q. Ngo (SUNY Buffalo)
Organizing Committee Chair	Atri Rudra (SUNY Buffalo)
Organizing Committee Members	Hung Q. Ngo (SUNY Buffalo) Sheng Zhong (SUNY Buffalo)

Technical Program Committee

Srinivas Aluru	Iowa State University, USA
Lars Arge	University of Aarhus, Denmark
Vikraman Arvind	Institute of Mathematical Sciences, India
James Aspnes	Yale University, USA
Mikhail Atallah	Purdue University, USA
Gill Barequet	Technion - Israel Institute of Technology, Israel
Michael Brudno	University of Toronto, Canada
Jianer Chen	Texas A&M University, USA
Bhaskar DasGupta	University of Illinois at Chicago, USA
Anupam Gupta	Carnegie Mellon University, USA
Lane A. Hemaspaandra	University of Rochester, USA
Kazuo Iwama	Kyoto University, Japan
Avner Magen	University of Toronto, Canada
Peter Bro Miltersen	University of Aarhus, Denmark
Hung Q. Ngo	SUNY Buffalo, USA, Chair
Mohammad Salavatipour	University of Alberta, Canada
Alan Selman	SUNY Buffalo, USA
Maria Serna	Universitat Politecnica de Catalunya, Spain
Hans Ulrich Simon	Ruhr University Bochum, Germany
Daniel Stefankovic	University of Rochester, USA
Chaitanya Swamy	University of Waterloo, Canada
My T. Thai	University of Florida, USA
Philipp Woelfel	University of Calgary, Canada

External Referees

Ashkan Aazami	Sariel Har-Peled	Chandan Saha
Oswin Aichholzer	Xin He	Rahul Santhanam
Gadi Aleksandrowicz	Patrick Healy	Abhinav Sarje
Andrei Asinowski	Lisa Higham	Srinivasa Rao Satti
Nikhil Bansal	Petr Hlineny	Pranab Sen
Siavosh Benabbas	Tuan Hoang	Jeffrey Shallit
Philip Bille	Falk Hoffner	Saad Sheikh
Christian Blum	John Iacono	Yaoyun Shi
Hans Bodlaender	Christian Igel	Akiyoshi Shioura
Vincenzo Bonifaci	Hiro Ito	David Shmoys
Prosenjit Bose	Riko Jacob	Anastasios Sidiropoulos
Andreas Brandstadt	Sanjay Jain	Preet Singh
Chad Brewbaker	Navdeep Jaitly	Michiel Smid
Josh Buresh-Oppenheim	Bharat Jayaraman	Jack Snoeyink
Zhipeng Cai	Pushkar Joglekar	Srikanth Srinivasan
Jin-Yi Cai	Allan G. Jørgensen	Clifford Stein
Cezar Campeanu	Matya Katz	Michal Stern
Paz Carmi	Telikepalli Kavitha	Lorna Stewart
Venkat. Chakaravarthy	Stephen Kobourov	Maxim Sviridenko
Deeparnab Chakrabarty	Johannes Koebler	Zoya Svitkina
Jason Corso	Jochen Konemann	Yasuhiro Takahashi
Brian Dean	Swastik Kopparty	Till Tantau
Josep Diaz	Guy Kortsarz	Dimitrios Thilikos
Martin Dietzfelbinger	Yang Liu	Iannis Turlakis
David Eppstein	Pinyan Lu	Kasturi Varadarajan
Jia-Hao Fan	Mohammad Mahdian	Juan Vera
Jiri Fiala	Thomas Mailund	Jacques Verstraete
Lance Fortnow	Kazuhisa Makino	Kira Vyatkina
Guillem Frances	Elvira Mayordomo	Stephan Waack
Gudmund S. Frandsen	Hartmut Messerschmidt	Andre Wehe
Hiroshi Fujiwara	Gabriel Moruz	Shmuel Wimer
Joaquim Gabarro	Jose Oncina	Mutsunori Yagiura
Ariel Gabizon	Rom Pinchasi	Xiao Yang
Qi Ge	Imran Pirwani	Sheng Zhong
Konstantinos Georgiou	J. Radhakrishnan	Jaroslav Zola
Tobias Glasmachers	Stanislaw Radziszowski	Anastasios Zouzias
Christian Glasser	Kenneth Regan	
Xiaoyang Gu	Christian Reitwiessner	

Table of Contents

Invited Talk

Bidding on Configurations in Internet Ad Auctions	1
<i>S. Muthukrishnan</i>	

1 Algorithmic Game Theory and Coding Theory

An Attacker-Defender Game for Honeynets	7
<i>Jin-Yi Cai, Vinod Yegneswaran, Chris Alfeld, and Paul Barford</i>	
On the Performances of Nash Equilibria in Isolation Games	17
<i>Vittorio Bilò, Michele Flammini, Gianpiero Monaco, and Luca Moscardelli</i>	
Limits to List Decoding Random Codes	27
<i>Atri Rudra</i>	

2 Algorithms and Data Structures

Algorithm for Finding k -Vertex Out-trees and Its Application to k -Internal Out-branching Problem	37
<i>Nathann Cohen, Fedor V. Fomin, Gregory Gutin, Eun Jung Kim, Saket Saurabh, and Anders Yeo</i>	
A $(4n - 4)$ -Bit Representation of a Rectangular Drawing or Floorplan	47
<i>Toshihiko Takahashi, Ryo Fujimaki, and Youhei Inoue</i>	
Relationship between Approximability and Request Structures in the Minimum Certificate Dispersal Problem	56
<i>Tomoko Izumi, Taisuke Izumi, Hirotaka Ono, and Koichi Wada</i>	

3 Graph Drawing

Coordinate Assignment for Cyclic Level Graphs	66
<i>Christian Bachmaier, Franz J. Brandenburg, Wolfgang Brunner, and Raymund Fülöp</i>	
Crossing-Optimal Acyclic HP-Completion for Outerplanar st -Digraphs	76
<i>Tamara Mchedlidze and Antonios Symvonis</i>	

Edge-Intersection Graphs of k -Bend Paths in Grids	86
<i>Therese Biedl and Michal Stern</i>	

4 Algorithms and Data Structures

Efficient Data Structures for the Orthogonal Range Successor Problem	96
<i>Chih-Chiang Yu, Wing-Kai Hon, and Bing-Feng Wang</i>	
Reconstruction of Interval Graphs	106
<i>Masashi Kiyomi, Toshiki Saitoh, and Ryuhei Uehara</i>	
A Fast Algorithm for Computing a Nearly Equitable Edge Coloring with Balanced Conditions	116
<i>Akiyoshi Shioura and Mutsunori Yagiura</i>	

5 Cryptography and Security

Minimal Assumptions and Round Complexity for Concurrent Zero-Knowledge in the Bare Public-Key Model	127
<i>Giovanni Di Crescenzo</i>	
Efficient Non-interactive Range Proof	138
<i>Tsz Hon Yuen, Qiong Huang, Yi Mu, Willy Susilo, Duncan S. Wong, and Guomin Yang</i>	
Approximation Algorithms for Key Management in Secure Multicast ...	148
<i>Agnes Chan, Rajmohan Rajaraman, Zhifeng Sun, and Feng Zhu</i>	

6 Algorithms

On Smoothed Analysis of Quicksort and Hoare's Find	158
<i>Mahmoud Fouz, Manfred Kufleitner, Bodo Manthey, and Nima Zeini Jahromi</i>	
On an Online Traveling Repairman Problem with Flowtimes: Worst-Case and Average-Case Analysis	168
<i>Axel Simroth and Alexander Souza</i>	
Three New Algorithms for Regular Language Enumeration	178
<i>Margareta Ackerman and Erkki Mäkinen</i>	

7 Computational Geometry

Convex Partitions with 2-Edge Connected Dual Graphs	192
<i>Marwan Al-Jubeih, Michael Hoffmann, Mashhood Ishaque, Diane L. Souvaine, and Csaba D. Tóth</i>	

The Closest Pair Problem under the Hamming Metric	205
<i>Kerui Min, Ming-Yang Kao, and Hong Zhu</i>	
Space Efficient Multi-dimensional Range Reporting	215
<i>Marek Karpinski and Yakov Nekrich</i>	

8 Approximation Algorithms

Approximation Algorithms for a Network Design Problem	225
<i>Binay Bhattacharya, Yuzhuang Hu, and Qiaosheng Shi</i>	
An FPTAS for the Minimum Total Weighted Tardiness Problem with a Fixed Number of Distinct Due Dates	238
<i>George Karakostas, Stavros G. Kolliopoulos, and Jing Wang</i>	
On the Hardness and Approximability of Planar Biconnectivity Augmentation	249
<i>Carsten Gutwenger, Petra Mutzel, and Bernd Zey</i>	

9 Computational Biology and Bioinformatics

Determination of Glycan Structure from Tandem Mass Spectra	258
<i>Sebastian Böcker, Birte Kehr, and Florian Rasche</i>	
On the Generalised Character Compatibility Problem for Non-branching Character Trees	268
<i>Ján Maňuch, Murray Patterson, and Arvind Gupta</i>	
Inferring Peptide Composition from Molecular Formulas	277
<i>Sebastian Böcker and Anton Pervukhin</i>	
Optimal Transitions for Targeted Protein Quantification: Best Conditioned Submatrix Selection	287
<i>Rastislav Šrámek, Bernd Fischer, Elias Vicari, and Peter Widmayer</i>	
Computing Bond Types in Molecule Graphs	297
<i>Sebastian Böcker, Quang B.A. Bui, Patrick Seeber, and Anke Truss</i>	

10 Sampling and Learning

On the Diaconis-Gangolli Markov Chain for Sampling Contingency Tables with Cell-Bounded Entries	307
<i>Ivona Bezáková, Nayantara Bhatnagar, and Dana Randall</i>	
Finding a Level Ideal of a Poset	317
<i>Shuji Kijima and Toshio Nemoto</i>	

A Polynomial-Time Perfect Sampler for the Q-Ising with a Vertex-Independent Noise	328
<i>M. Yamamoto, S. Kijima, and Y. Matsui</i>	
Extracting Computational Entropy and Learning Noisy Linear Functions	338
<i>Chia-Jung Lee, Chi-Jen Lu, and Shi-Chun Tsai</i>	
HITS Can Converge Slowly, but Not Too Slowly, in Score and Rank	348
<i>Enoch Peserico and Luca Pretto</i>	

11 Algorithms

Online Tree Node Assignment with Resource Augmentation	358
<i>Joseph Wun-Tat Chan, Francis Y.L. Chin, Hing-Fung Ting, and Yong Zhang</i>	
Why Locally-Fair Maximal Flows in Client-Server Networks Perform Well.....	368
<i>Kenneth A. Berman and Chad Yoshikawa</i>	
On Finding Small 2-Generating Sets	378
<i>Isabelle Fagnot, Guillaume Fertin, and Stéphane Vialette</i>	
Convex Recoloring Revisited: Complexity and Exact Algorithms	388
<i>Iyad A. Kanj and Dieter Kratsch</i>	
Strongly Chordal and Chordal Bipartite Graphs Are Sandwich Monotone	398
<i>Pinar Heggernes, Federico Mancini, Charis Papadopoulos, and R. Sriharan</i>	

12 Complexity and Computability

Hierarchies and Characterizations of Stateless Multicounter Machines...	408
<i>Oscar H. Ibarra and Ömer Eğecioğlu</i>	
Efficient Universal Quantum Circuits	418
<i>Debajyoti Bera, Stephen Fenner, Frederic Green, and Steve Homer</i>	
An Improved Time-Space Lower Bound for Tautologies	429
<i>Scott Diehl, Dieter van Melkebeek, and Ryan Williams</i>	

13 Probabilistic Analysis

Multiple Round Random Ball Placement: Power of Second Chance	439
<i>Xiang-Yang Li, Yajun Wang, and Wangsen Feng</i>	

The Weighted Coupon Collector's Problem and Applications	449
<i>Petra Berenbrink and Thomas Sauerwald</i>	
Sublinear-Time Algorithms for Tournament Graphs	459
<i>Stefan Dantchev, Tom Friedetzky, and Lars Nagel</i>	

14 Complexity and Computability

Classification of a Class of Counting Problems Using Holographic Reductions	472
<i>Michael Kowalczyk</i>	
Separating NE from Some Nonuniform Nondeterministic Complexity Classes	486
<i>Bin Fu, Angsheng Li, and Liyu Zhang</i>	
On the Readability of Monotone Boolean Formulae	496
<i>Khaled Elbassioni, Kazuhisa Makino, and Imran Rauf</i>	

15 Algorithms and Data Structures

Popular Matchings: Structure and Algorithms	506
<i>Eric McDermid and Robert W. Irving</i>	
Graph-Based Data Clustering with Overlaps	516
<i>Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann</i>	
Directional Geometric Routing on Mobile Ad Hoc Networks	527
<i>Kazushige Sato and Takeshi Tokuyama</i>	
Author Index	539

Bidding on Configurations in Internet Ad Auctions

S. Muthukrishnan

Google Inc., 76 9th Av, 4th Fl., New York, NY, 10011
muthu@google.com

Abstract. In Internet advertising, a configuration of ads is determined by the seller, and advertisers buy spaces in the configuration. In this paper, motivated by sponsored search ads, we propose an auction where advertisers directly bid and determine the eventual configuration.

1 Motivation

In Internet advertising, a *configuration* of ads is determined by the seller, and advertisers buy spaces within the configuration. For example, a publisher like nytimes.com may set aside space at the top right corner for display of image or video ads, and space on the lower right corner for text ads. Advertisers then pay to show their ads in these spaces. In sponsored search, search engines set aside space on the right side of the page (and in some cases, at the top or even the bottom) for text ads. Advertisers take part in an auction that determines the arrangement of ads shown in these spaces.

The configuration chosen by the seller is designed to make the ads effective. Publishers have insights from marketing and sociological studies on effective placements and viewer satisfaction. Similarly, advertisers have their own insights and can influence the publishers directly by collaborating on the design of configurations or influence indirectly by buying spaces at different rates.

We study how the choice of the configuration can be determined directly by letting parties bid on the configuration. A configuration depends on not only where the space is allocated, but also, how much space is allocated, what type of ads are allowed, how many ads fit into a space, how the ads are formatted in font, color and size, etc. Thus the bidding language may become complex. In addition, effectiveness of configurations also depends on viewer behavior and response which has to be modeled. Thus, determining optimal or efficient auctions become sophisticated optimization problems. Finally, the game theory of bidding on configurations becomes nuanced. Therefore, study of ad configuration problems and associated mechanism design has the potential to become a rich research area in the future.

In this context, we formulate a very simple problem motivated by sponsored search. In particular, we let advertisers directly impact the *number* of ads shown. We present an auction that generalizes the current auctions for sponsored search, and show that it can be implemented in near-linear time.

2 Our Problem

In sponsored search, we have a set of advertisers who bid on keywords. When a user poses a search query, an auction is run among all advertisers who bid on keywords relevant to the query. The advertisers are sorted say based on their bids and the top ℓ are shown in the decreasing order of their bids, charging each advertiser only the bid of the ad below. This is called the *Generalized Second Price* auction. Search engines use this auction in different forms, for example, by sorting based on other criteria, adding reserve prices, etc. See [4,3,6,9] for more details.

In this application, we refer to the ℓ positions arranged from top to bottom as the *configuration*. This focuses our attention on the simple aspect of the configuration, namely, the number ℓ of ad positions in it. In the problem we study, we make this biddable.

Precisely, our problem is as follows. We have n advertisers; i th advertiser has bid b_i , the maximum they are willing to pay for an ad, private value v_i , as well as p_i , that represents the condition that the advertiser will not like to appear in any auction that results in more than p_i positions. Thus, this model lets advertisers directly bid on the number of positions in the resulting configuration of ads shown. For example, if an advertiser i sets $p_i = 1$, they wish to be shown only if they are the exclusive (or solitary) ad. If every advertiser sets $p_i \geq \ell$ where ℓ is the maximum number of positions in the configuration, then this scenario becomes the standard ad auction described above with ℓ positions.

The problem now is to design a suitable auction, that is, the output will comprise

- the number $k^* \leq n$ of ads that will be shown,
- the arrangement of k^* ads to show, and
- a pricing for each ad.

3 Our Solution

We propose a mechanism that is in many ways similar to current ad auctions. This is described in [3,5,6] and we use insights and concepts from there.

Our algorithm works in two steps as follows.

- For each k , where k denotes the number of ads shown, we determine the maximum value assignment, assuming of course that $b_i = v_i$. This will give us the choice of k^* .
- Then, we charge each advertiser the minimum bid they should have made (fixing all others) in order to get the assignment they obtained.

There are details to both steps.

Finding optimal value configuration. For each k , define $V(k)$ to be maximum value assignment of ads to at most k positions. Then, we define $k^* = \operatorname{argmax}(V(k))$. Renumber the advertisers so they are numbered in the increasing order of their bids. Thus, b_i is the i th largest bid, and corresponds now to advertiser renumbered to i .

Lemma 1. Define i_k^* to be the smallest i such that $|\{j | j \leq i, p_j \geq k\}| = k$, if it exists. Then,

$$V(k) = \sum_{j \leq i_k^*, p_j \geq k} b_j.$$

Proof. The maximum value assignment is simply the maximum bipartite matching of eligible ads to positions. ■

Our algorithm is simple. We represent each advertiser i by a two dimensional point (i, p_i) with weight b_i . Then, we can determine i_k^* by binary searching on i with range query that counts the number of points in $[0, i] \times [k, \infty]$. And we can determine $V(k)$ by a single range sum query on $[0, i_k^*] \times [k, \infty]$. Both of these can be answered in $O(\log n)$ time using range searching data structures [7]. Thus the total running time of the algorithm is $O(n \log n)$ (to sort the bids) + $O(n \log^2 n)$ (for each $k = 1, \dots, n$, we do $O(\log n)$ binary search queries each of which takes $O(\log n)$ time); hence, the total time complexity is $O(n \log^2 n)$.

Theorem 1. There is an $O(n \log^2 n)$ time algorithm to compute k^* and produce an arrangement of ads in k^* slots with maximum value $V(k^*)$.

A natural question is if $V(k)$ is a single mode function, increasing as k increases from 1 and decreasing beyond the maximum at some k^* . If that were the case, the search for k^* would be much more efficient. Unfortunately, that is not true. Consider a sequence of (b, p) pairs as follows:

$(1000, 1), (400, 3), (400, 3), (400, 3), (300, 5), (300, 5), (300, 5), (290, 5), (100, 5), \dots$

which produces a see-saw teeth of $V(k)$'s: $V(1) = 1000$, $V(2) = 800$, $V(3) = 1200$, $V(4) = 1190$, etc. We can construct examples with linear number of teeth of maxima if needed.

Another natural question is if $V(k^*)$ can be computed by simply auctioning one position after another, say going top down, as in [8]. This is unfortunately incorrect since without fixing a k *a priori*, it is impossible to determine the advertisers who compete for any position.

Pricing. We now design an algorithm to calculate a price for each advertiser; it will have the *minimum pay property* (mpp), that is, each advertiser would pay no more than what he would have bid, if he knew all others' bids, to get the exact assignment he got. More formally, say the solution above returns a configuration of k^* ads and an advertiser i has position j in it. We want to charge advertiser i , the minimum bid c_i he would make, if all other bids were fixed, to get that position j in a configuration with $k^* - 1$ other ads. In other words, c_i should not only be sufficient to get position j in the current configuration, but also should ensure that the outcome of the previous step, ie., the optimal configuration, has k^* positions altogether.

Determining prices that satisfy mpp is trivial in the absence of p_i constraints: advertiser i in position j would simply pay the bid of the ad in position $j + 1$

plus ε , giving the *generalized second price* (GSP) auction that is currently popular [3,5,6].¹ In our case, determining mpp price is not trivial.

We first show a number of natural properties of mpp pricing from GSP auctions that no longer hold for our case.

- There are cases when mpp price for advertiser i is not the bid of the advertiser below.

Consider optimal allocation with $k = 2$ of $(100, 50)$ and that with $k = 3$ of $(100, 30, 22)$ because the advertiser with bid 50 does not want to appear with 3 or more positions. Clearly $k = 3$ has the optimal value. Then, if 30 bid $22 + \varepsilon$, he would still get his position in the $k = 3$ configuration, but the $k = 2$ configuration is now the optimal one.

- The mpp price for advertiser i may not be the maximum of the mpp bids in each of the auction outcomes in which i appears.

The example above works, since we can fix it so that 30 appears only on $k = 3$ case.

- The mpp price for advertiser i may not be anyone's bid (or $\pm \varepsilon$ of it).

In the example above, the mpp price for advertiser 30 is $28 + \varepsilon$.

In what follows, we present an efficient algorithm to compute the mpp prices.

Definition 1. Let C_i^* define the optimal configuration of advertisements with i positions, and let C_{-i}^* define the optimal configuration for i positions, but include the $i + 1$ qualifying bidder as well.² Define b to be the mpp price for i and let $b_{\downarrow i}^j$ be the maximum bid below b_i in C_j^* for any j . Define $V(C)$ to be the value in C , that is $V(C) = \sum_{i \in C} b_i$.

We make a few observations.

Observation 1. For each advertiser i , there exist α_i, β_i , $1 \leq \alpha_i \leq \beta_i \leq n$ such that i appears in every one of configurations $C_{\alpha_i}^*, \dots, C_{\beta_i}^*$, and none other.

Say $C_{k^*}^*$ is the optimal configuration and i appears in it.

Observation 2. We have $b > b_{\downarrow i}^{k^*}$.

We divide $C_1^*, \dots, C_n^*$ into three categories and use structural properties in each category.

Consider $\beta_i \geq j > k^*$, if any. We have,

Lemma 2. $b_{\downarrow i}^j < b_{\downarrow i}^{k^*} < b$.

Consider any j such that C_j^* does not contain i . Then,

¹ The ε is a little extra, like say one cent, that advertisers are charged to “beat” the bid below.

² We assume such a bidder exists. The remaining arguments can be easily modified otherwise.

Lemma 3. $b \geq \max_j \{V(C_j^*) - V(C_{k^*}^*) + b_i\}$.

The lower bound above on b can be computed from $\max_{j < \alpha_i} V(C_j^*)$ and $\max_{j > \beta_i} V(C_j^*)$.

Consider any $\alpha_i \leq j < k^*$. If decreasing i 's bid from b_i to b' does not change the set of advertisers in C_j^* , then $V(C_j^*) - b_i + b' < V(C_{k^*}^*) - b_i + b'$. Otherwise,

Lemma 4. $b \geq \max_j \{V(C_{-j}^*) - b_i\}$.

The lower bound above on b can be computed from $\max_{\alpha_i \leq j < k^*} V(C_{-j}^*)$.

Thus, we can now compute b to be the max of the two quantities above, plus ε ; we can also easily see that the resulting term will be no larger than b_i , and therefore a legal price. This gives,

Theorem 2. *The mpp price for any advertiser i can be computed in $O(\log n)$ time. Over all advertisers, price computations take $O(n \log n)$ time.*

For comparison, one might wish to consider Vickrey-Clarke-Groves (VCG) pricing [5,1,2]. Consider any i in $C_{k^*}^*$. It's VCG price can be seen to be

$$V(C_{k^*}^*) - b_i - \max \left\{ \max_{\ell < \alpha_i} V(C_\ell^*), \max_{\alpha_i \leq \ell \leq \beta_i} V(C_{-\ell}^*), \max_{\beta_i < \ell} V(C_\ell^*) \right\},$$

which can again be computed in $O(\log n)$ time.

Remark. Say there is an upper bound ℓ on the number of slots that the bidders are allowed to specify, as determined by the auctioneer (the search company). Then our auction can be implemented in $O(n\ell)$ time easily by considering each potential k of total slots from 1 to ℓ . However the implementation we have described above is more efficient for large ℓ (note that in theory, ℓ may be as high as n).

4 Concluding Remarks

We formulated a very simple ad configuration auction problem where advertisers directly bid on the number of positions in the configuration. We proposed an auction and showed a near-linear time algorithm to compute the minimum pay property (mpp) price as well as the VCG price. Of immediate interest is the equilibrium properties of our auction. When $p_i \geq 1$ for all i , our auction becomes the standard second price (VCG) auction and hence it is truthful. When $p_i \geq n$ for all i , our auction becomes the standard generalized second price (GSP) auction which is not truthful, but is known to have an equilibrium identical to the VCG auction [3,5,6]. Does our auction have such good equilibriums and are there natural dynamics that will let advertisers converge to the good equilibrium? Adopting the arguments in [3,6] might help address this question.

More broadly, we hope mechanism design and game theory of ad configuration problems get studied in more detail. In particular, what are suitable bidding languages for configurations, what are the relative powers of different bidding

languages, what are suitable auctions that are easy to understand, implement and have desirable efficiency, revenue properties, what are achievable equilibrium properties and associated dynamics, etc.

Acknowledgements

We thank ad colleagues at Google who provide a stimulating environment.

References

1. Clarke, E.: Multipart pricing of public goods. *Public Choice* 11, 17–33 (1971)
2. Groves, T.: Incentives in teams. *Econometrica* 41(4), 617–631 (1973)
3. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. In: *Second workshop on sponsored search auctions* (2006)
4. Varian, H.: Position auctions. *International Journal of Industrial Organization* 25(6), 1163–1178 (2007)
5. Vickrey, W.: Counterspeculation, auctions and competitive-sealed tenders. *Finance* 16(1), 8–37 (1961)
6. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: *ACM Conference on Electronic Commerce, EC* (2006)
7. Agarwal, P., Erickson, J.: Geometric range searching and its relatives. In: *Advances in Discrete and Computational Geometry 1999*, pp. 1–56. American Mathematical Society (1999)
8. Aggarwal, G., Feldman, J., Muthukrishnan, S.M.: Bidding to the top: VCG and equilibria of position-based auctions. In: Erlebach, T., Kaklamanis, C. (eds.) *WAOA 2006*. LNCS, vol. 4368, pp. 15–28. Springer, Heidelberg (2007)
9. Muthukrishnan, S.: Internet Ad Auctions: Insights and Directions. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I*. LNCS, vol. 5125, pp. 14–23. Springer, Heidelberg (2008)

An Attacker-Defender Game for Honeynets

Jin-Yi Cai¹, Vinod Yegneswaran², Chris Alfeld³, and Paul Barford^{1,3}

¹ University of Wisconsin, Madison, WI

² SRI International, Menlo Park, CA

³ Nemean Networks, Madison, WI

jyc@cs.wisc.edu, vinod@csl.sri.com, alfeld@nemeannetworks.com,
pb@cs.wisc.edu

Abstract. A honeynet is a portion of routed but otherwise unused address space that is instrumented for network traffic monitoring. It is an invaluable tool for understanding unwanted Internet traffic and malicious attacks. We formalize the problem of defending honeynets from systematic mapping (a serious threat to their viability) as a simple two-person game. The objective of the *Attacker* is to identify a honeynet with a minimum number of probes. The objective of the *Defender* is to maintain a honeynet for as long as possible before moving it to a new location within a larger address space. Using this game theoretic framework, we describe and prove optimal or near-optimal strategies for both Attacker and Defender. This is the first mathematically rigorous study of this increasingly important problem on honeynet defense. Our theoretical ideas provide the first formalism of the honeynet monitoring problem, illustrate the viability of network address shuffling, and inform the design of next generation honeynet defense.

1 Introduction

Malicious activity in the Internet is a significant and growing problem. The spectrum of threats includes denial-of-service (DoS) attacks, self-propagating worms, spam, spyware, and botnets. Malicious parties (Black Hats) are constantly looking for new potential victim systems, which are typically identified by scanning across large portions of the Internet address space. Many tools have been developed to facilitate this task, and researchers describe and evaluate the magnitude of the threat posed by highly efficient scanning methods (*e.g.*, [9]).

The network security community has recognized that the process of scanning for victims actually offers a unique opportunity to track malicious activity in the Internet. Standard configurations for destination networks typically drop packets destined for addresses that are not associated with a live system. Changing this configuration to route packets to a monitoring system enables passive collection of scanning packets. Even more compelling is the notion of routing these packets to systems with the ability to engage in conversations with the malicious hosts, thereby enabling a detailed attack profile to be gathered. This configuration is commonly referred to as a *honeynet* (*i.e.*, a network of honeypots).

Over the past several years, many honeynet installations have emerged across the globe, which have been used to monitor and identify many important details of malicious activity, such as the characterization studies of Internet “background radiation” [5], large worm outbreaks [3], and botnet activity [4]. Furthermore, research on

honeynet systems is active and focused on enabling scalable and secure measurement at greater levels of detail.

The successes and utility of honeynets are not likely to be lost on the Black Hats. Several recent studies have shown that well-known monitoring entities such as Dshield.org [10] can be identified effectively by using different probing methods [1,8,7]. Black Hats can adopt these techniques to create blacklists of address space for honeynets, which would effectively render them useless.

In this paper, we address the problem of how to thwart attempts to map honeynets. To the best of our knowledge, we are the first to address this problem formally (it is mentioned briefly in [11]). We model the situation as a two-player game between an Attacker and a Defender. The objective of the Attacker is to identify the segment of monitored addresses (*i.e.*, the honeynet) within a larger address space. The Attacker does this by sending probes to the address space. The responses typically reveal distinguishing characteristics of the type of addresses being probed, *i.e.*, corresponding with either a live system or an address within the honeynet. An important capability in some honeynets is that they can respond to probes in a protocol compliant way (*e.g.*, [13]). When a probe is received on a honeynet address, the Defender can obfuscate its response by mimicking what would have been sent by a live system on a regular IP address. However, this obfuscation takes resources, and for a given resource bound, the Attacker will eventually be able to distinguish those responses coming from a honeynet.

The objective of a honeynet is to collect data as long as possible without being mapped by an Attacker. However, once identified, a honeynet will have to be assigned to a new set of addresses. This remapping or *shuffling* is a costly process. An additional objective of the Attacker is to identify the honeynet with a *minimum number of probes*. The Defender has two objectives. The first is to prevent the honeynet from being identified, which is done by periodically shuffling its location within the address space. The second objective is to extend the duration of shuffling epochs in order to minimize demands on the system responsible for shuffling. The Defender can use its protocol mimicking capability as a means for delaying the reshuffling. The game itself will focus on what transpires between shuffling epochs.

Mathematically, we model the response by the Defender for any probe as either 0 or 1. If the probe is sent to an IP address that is not part of the honeynet, then the response is 0. If the probe is sent to an IP address that is part of the honeynet, then the Defender may either respond truthfully with 1, or obfuscate with a “lie” 0. We assume that the Attacker cannot distinguish between a lie and the response from a live host. We model resource limitations as a *global lie budget* for the Defender. Within this context, we are able to prove optimal or near-optimal strategies for both the Attacker and Defender.

In Section 2.1, we first formally define the game. Then we develop general bounds, and the analysis leads to two particular strategies. The main theorem is Theorem 2, which establishes a *unique optimality* result. The proof is based on a key combinatorial lemma (Lemma 1) about certain optimal packing strategies. This lemma is easy to understand and quite plausible intuitively, but, its proof is more difficult. We then extend our basic formulation of the game to consider the situation of having multiple segments of honeynets within the address space. Here we introduce a strategy with an amortized complexity analysis that is optimal up to a constant factor.

Our basic game theoretic formulation plus the extensions enable us to analyze the processes of attacking and defending honeynets experimentally. We conducted a series of evaluations over a range of possible configurations to assess the trade-offs between address space size, honeynet size, probe rates, and shuffling frequency. In [12], we discuss practical considerations involved in network address shuffling and evaluate our implementation of a network address shuffling middlebox called Kaleidoscope. Due to space limitations, some details and proofs are omitted (see our techreport [2]).

2 The Attacker-Defender Game

2.1 Basic Formulation

We model the adversary/honeynet interaction within a reshuffling epoch as a 2-person game between an Attacker and a Defender. A contiguous segment of monitored addresses is placed randomly within an address space. We call monitored addresses (*i.e.*, the honeynet) “black” and all other addresses “white”. During the game the Attacker queries addresses and receives a reply based on the color of the address queried. White addresses must reply “white” (represented by a bit 0) but black addresses may answer black (represented by a bit 1) or “lie” and answer “white” (a bit 0). We impose a global limit on the number of lies allowed but they can be used flexibly. This global limit is imposed to reflect the cost on total system memory and other resources (within a reshuffling epoch). (See technical report for more discussions on the model and its variations.)

Formally, let m, k and ℓ be positive integers. Let $n = mk$. Our address space is a circular array of n cells, identified with the additive group $\mathbf{Z}_n$, *i.e.*, each cell is indexed by some $i \in \mathbf{Z}_n$. We use the following notation:

Definition 1. A block is a contiguous subset of $\mathbf{Z}_n$ of size m . Denote by $B_j = \{j, j+1, \dots, j+m-1\}$ (indices are mod n), the block that starts at the j -th cell. As an alternate notation, $B^j = \{j-m+1, \dots, j-1, j\}$ denotes the block that ends at j . The honeynet monitors are identified with a single block of black addresses B^c that ends at some c .

The game starts with a random spin of B^c , that is, a uniformly distributed $c \in \mathbf{Z}_n$. We assume that both players know the parameters m, k , and ℓ . The position c is known to the Defender D but unknown to the Attacker A whose aim is to find it. In a round of the game (*i.e.*, between shuffling epochs), A moves first and the two players alternate their moves until A knows c . A move of A consists of a *query* at a cell j . If $j \notin B^c$ then D replies with the bit $b = 0$. If $j \in B^c$ then D has a choice: D can either answer $b = 1$ or, *commit a lie* by answering $b = 0$. However, D can only lie $< \ell$ times in total. In short, n is the size of the circular address space, m is the length of a monitor block, $k = n/m$, and ℓ is the *quota* of lies. (Note that $\ell = 1$ is the smallest possible and represents no lies allowed. We also note that the lie quota should only be considered a fixed parameter for the duration between reshuffling epochs. The assumption that an attacker knows the block size is in keeping with the tradition in security where we want to err on the side of giving attackers extra information. This information of block size is hard to keep as a secret at any rate, and one can reasonably expect that an estimate by attackers can be obtained with some probes).

Denote by A (respectively D) a strategy by the Attacker (respectively the Defender). We will primarily analyze *pure* (nonrandomized) strategies; but with a slight abuse of notation we will use the same notation for randomized strategies. Let $V = V(A, D)$ denote the number of queries A makes against D until A learns the value c . Thus, V is a variable randomized over the uniform choices of $c \in \mathbf{Z}_n$ (and possibly over the randomization of the randomized strategies of the two players). Let v denote the expectation $v = v(A, D) = \mathbf{E}[V(A, D)]$. The objective of the Attacker is to minimize this quantity v . The objective of the Defender is to maximize v .

2.2 The Strategies: Round-Robin (RR) and Delay-Delay (DD)

Before giving a more detailed analysis of this game, we make some general observations. Suppose, during a round of the game, A learns a cell $j \in B^c$. If $m = 1$, then $c = j$ and the game is over. If $m > 1$, then, because B^c occupies a contiguous segment of cells, A can be sure that B^c is located in one of the following m possible ways (addition is done in $\mathbf{Z}_n$): $c = j, j + 1, \dots, j + m - 1$. Then A can “zero-in” on the boundaries of B^c by performing the following binary search. First, A queries the cells $j^- = j - \lfloor m/2 \rfloor$ and $j^+ = j + \lfloor m/2 \rfloor$. At least one of j^- and j^+ must belong to B^c because the number of cells located strictly in between these two cells is $2\lfloor m/2 \rfloor - 1 < m$. Thus, D must either choose to answer truthfully $b = 1$ at least once, or (if permissible by the quota) commit an additional lie. If A receives both answers with $b = 0$, then he knows that a lie has been committed. So A continues to query j^+ and j^- until D answers $b = 1$. Assume D answers $b = 1$ to the query j^+ ; the case for j^- is symmetric. At this point, A knows that c is among the following $\lceil m/2 \rceil$ locations: $c = j^+, j^+ + 1, \dots, j + m - 1$. So A can effectively consider a “shrunk” version of the problem as follows: Identify the cells between j and j^+ as a single cell. This identification effectively reduces the length of the block by $\lfloor m/2 \rfloor$ to $\lceil m/2 \rceil$. Now we can consider the problem of placing the shortened block on the condition that it contains a certain cell (the identified j and j^+). This describes a “binary search” process.

It is clear that every step of this “binary search” reduces the length from m' to $\lceil m'/2 \rceil$. When the length has shrunk to 1, the game is over. If $2^{r-1} < m \leq 2^r$, then it takes exactly $r = \lceil \log_2 m \rceil$ steps to reduce the length to 1. If A queries j^- and j^+ in random order, then on average each step takes 1.5 queries. Let ℓ' be the number of lies D is still allowed to commit ($\ell' < \ell$ and may be much less because of lies committed prior to the “binary search”). Then we have an upper bound of $2\ell' + 1.5\lceil \log_2 m \rceil$ queries under this Attacker strategy, valid for any D .

We see that once the Attacker gains the knowledge of one $j \in B^c$, A can zero-in fairly rapidly. Therefore, it is reasonable to assume that a rational Defender will not reveal any $j \in B^c$ until being forced to (this statement will be qualified, see below). This reasoning leads us to the following Delay-Delay (DD) strategy for the Defender.

Definition 2. *The strategy Delay-Delay (DD) is as follows. Lie as long as the quota of lies allows.*

Definition 3. *The strategy Round-Robin (RR) is as follows. Pick any cell to start, e.g., $j = 0$. Query j consecutively ℓ times. Then set $j := j + m$, and repeat. As soon as a reply of $b = 1$ is received switch to a “binary search” strategy to zero-in.*

Note that with at most $k\ell$ queries, RR will obtain a $b = 1$ answer, and commence its “binary search”. Indeed, when RR has made $k\ell$ queries, these would have been for each $j = im$ for $0 \leq i < k$ and each such j is queried exactly ℓ times. For any c , some query j will be in B^c (in particular, for $i = \lfloor c/m \rfloor$). For any D , up to ℓ repeated queries at this j must yield an answer $b = 1$. On average, RR will take $\frac{1}{k} \sum_{i=1}^k i\ell = (k+1)\ell/2$ queries against DD to arrive at its First-Time T . Therefore, $V(\text{RR}, \text{DD}) \leq k\ell + 2\lceil \log_2 m \rceil$, in the worst case, and $(k+1)\ell/2 + \lceil \log_2 m \rceil \leq v(\text{RR}, \text{DD}) \leq (k+1)\ell/2 + 1.5\lceil \log_2 m \rceil$ on average. And against any other D , $v(\text{RR}, D) \leq k\ell/2 + 1 + 1.5\lceil \log_2 m \rceil + 2(\ell - 1)$.

2.3 Delay-Delay against Any Attacker

We provide a simple lower bound for $v(A, \text{DD})$. These considerations justify our adoption of DD for the Defender in the following. (Curiously, if the Defender knew that the Attacker is RR, then he actually should do the opposite of DD, and save his entire quota of lies for the “binary search” stage, saving a constant factor.) We note that (1) DD performs well against any A , and (2) for one particular A , namely $A = \text{RR}$, DD performs almost as well as any D (Section 2.2).

Definition 4. For an Attacker A and Defender D , the Capitulation-Time, $L = L(A, D)$, is the first time (the number of queries made) when A has made ℓ queries in B^c against D .

For an arbitrary pair (A, D) it is possible that the game ends (when A learns c) before A ever hits ℓ times in B^c . In this case we define $L = \infty$.

We can prove the following

Theorem 1.

$$\mathbf{E}[L(A, \text{DD})] \geq (\lfloor k\ell/2 \rfloor + 1)/2 > k\ell/4.$$

Note that our bound on $\mathbf{E}[L(A, \text{DD})]$ is within a factor 2 of the case when the Attacker uses RR against any Defender strategy D . Once the Attacker is at Capitulation-Time, all that remains is to pinpoint c . Doing so takes $\Omega(\log m)$ time. Thus we arrive at the following corollary.

Corollary 1. $v(A, \text{DD}) > k\ell/4 + \Omega(\log m)$.

3 Round-Robin Is optimal against Delay-Delay

In this section we establish our main result that RR is optimal against DD. In fact, we will prove a stronger theorem of *unique optimality* (Theorem 2).

Definition 5. An Attacker strategy A is essentially Round-Robin (*eRR*) if it is of the following form. The strategy A first makes ℓ queries at some cell j . It then updates j to j' where $j' \equiv j \pmod{m}$ and j' has not been queried and makes ℓ queries at j' . The strategy repeats this behavior until it finds some $j \in B^c$, i.e., receives $b = 1$, at which point it switches to a “binary-search”.

Clearly,

$$\mathbf{E}[L(\text{eRR}, \text{DD})] = \mathbf{E}[L(\text{RR}, \text{DD})]. \quad (1)$$

Theorem 2. *For any Attacker A that is not eRR ,*

$$\mathbf{E}[L(RR, DD)] < \mathbf{E}[L(A, DD)]. \quad (2)$$

Lemma 1. *Let m, k, p and ℓ be positive integers, and $n = mk$. Let S be a multi-set $\{S^1, S^2, \dots, S^p\}$, where each $S^i \subset \mathbf{Z}_n$ is a contiguous segment with $|S^i| = m$. We say $c \in \mathbf{Z}_n$ is **heavy** with respect to S if c is covered by at least ℓ blocks $S^i \in S$. Then,*

$$|\{c \in \mathbf{Z}_n \mid c \text{ is heavy}\}| \leq m \cdot \lfloor \frac{p}{\ell} \rfloor.$$

Let $C(S) = |\{c \in \mathbf{Z}_n \mid c \text{ is heavy}\}|$ denote the number of heavy points w.r.t. S . A weaker bound $C(S) \leq \lfloor m \cdot \frac{p}{\ell} \rfloor$ follows easily from a volume argument. But we need the stronger version to prove Theorem 2. The proof of the lemma is presented in the extended technical report version. Here we prove the Theorem assuming the Lemma.

Proof (Proof of Theorem 2). Clearly $\Pr[L(RR, DD) = \infty] = 0$. If $\Pr[L(A, DD) = \infty] \neq 0$, then (2) is obvious. Suppose $\Pr[L(A, DD) = \infty] = 0$. We are concerned with

$$\mathbf{E}[L] = \sum_{i=1}^{\infty} \Pr[L \geq i] = 1 + \sum_{i=1}^{\infty} \Pr[L > i], \quad (3)$$

for $L = L(A, DD)$ and $L(RR, DD)$. Our goal is to show that RR minimizes $\mathbf{E}[L]$. Let $\mathbf{E}_A[L] = \mathbf{E}[L(A, DD)]$ denote the expectation of the Capitulation-Time for Attacker A against Defender DD . Similarly, $\Pr_A$ indicates the probability for Attacker A against Defender DD . Our first goal is to prove the (nonstrict) dominance of RR :

$$\mathbf{E}_A[L] \geq \mathbf{E}_{RR}[L]. \quad (4)$$

We establish $\mathbf{E}_A[L] \geq \mathbf{E}_{RR}[L]$ by proving term by term $\Pr_A[L > i] \geq \Pr_{RR}[L > i]$ in the sum (3). Observe that if $i \geq k\ell$, then $\Pr_{RR}[L > i] = 0$. Thus, we only need to consider $i < k\ell$. The inequality is equivalent to $\Pr_A[L \leq i] \leq \Pr_{RR}[L \leq i]$. Define H_i to be the number of hits B^c received among the first i queries. Then the event $[L \leq i]$ is equivalent to $[H_i \geq \ell]$. Thus, we seek to show for all i ,

$$\Pr_A[H_i \geq \ell] \leq \Pr_{RR}[H_i \geq \ell]. \quad (5)$$

Imagine that we are given the first i (not necessarily distinct) query points. Each query point j defines a block B_j . We observe that j hits B^d iff d belongs to the block B_j . Let $\mathcal{S}_i$ be the configuration consisting of i blocks corresponding to the queries the Attacker would make assuming the Defender answers $b = 0$ to the first $i - 1$ queries. We claim $\Pr_A[H_i \geq \ell] = \frac{C(\mathcal{S}_i)}{n}$.

The equality is clear if all first i answers are indeed $b = 0$. We prove the equality is valid for the actual interaction that defines H_i . We prove this by induction. For $i = 1$ the result holds. Assume the result holds up to $< i$. The probability $\Pr_A[H_i \geq \ell]$ is $1/n$ times the number of d such that B^d was hit at least ℓ times during the first i queries.

$$\Pr_A[H_i \geq \ell] = \Pr_A[H_{i-1} \geq \ell] + \Pr_A[(H_i = \ell) \wedge (H_{i-1} < \ell)]. \quad (6)$$

By induction $\Pr_A[H_{i-1} \geq \ell] = \frac{C(S_{i-1})}{n}$. Now, for the second term, the conjunction $(H_{i-1} < \ell)$ implies that DD answers the first $i-1$ queries with $b=0$. Thus, the probability $\Pr_A[(H_i = \ell) \wedge (H_{i-1} < \ell)]$ counts the number of heavy points $d \in \mathbf{Z}_n$ that are heavy in S_i but not heavy in S_{i-1} . It follows that the sum of these two probabilities is exactly $\frac{C(S_i)}{n}$, completing the induction.

By Lemma 1, the probability $\Pr_A[H_i \geq \ell]$ is maximized by RR. Therefore, by the dominance of RR, term by term in (3), we obtain $\mathbf{E}_{\text{RR}}[L] \leq \mathbf{E}_A[L]$.

To prove the strict dominance of RR (and eRR) over any other A against DD, we reason as follows. If the first ℓ queries are not at the same cell, then at the ℓ -th query, RR produces exactly m heavy points while A has strictly less. Thus, at the ℓ -th term in the sum for $\mathbf{E}_A[L]$, the inequality $\Pr_A[H_\ell \geq \ell] < \Pr_{\text{RR}}[H_\ell \geq \ell]$ is strict. As we have the (nonstrict) dominance of RR for every term we arrive at $\mathbf{E}_{\text{RR}}[L] < \mathbf{E}_A[L]$.

We now assume that the first ℓ queries are at a single cell. Let j_1 be that location. If the next ℓ queries are not at some cell j_2 then consider the time step 2ℓ . By the same argument, at 2ℓ we have a strict inequality and a (nonstrict) dominance of RR elsewhere, that again gives $\mathbf{E}_{\text{RR}}[L] < \mathbf{E}_A[L]$. This argument proves that to be optimal, the locations of the queries $j_1, j_2, \dots$ must be repeated ℓ times each in succession.

Finally, consider the possibility of two query locations j and j' with $j \not\equiv j' \pmod{m}$. Then at time step $i = k\ell$, RR produces a *perfect* cover of all $\mathbf{Z}_n$ with multiplicity ℓ . Meanwhile, A has an *imperfect* cover of all $\mathbf{Z}_n$ that produces a strict inequality in favor of RR. So again, we find that $\mathbf{E}_{\text{RR}}[L] < \mathbf{E}_A[L]$.

This proves the strict optimality of RR (and eRR).

4 Extension to Multiple Monitoring Blocks

We briefly discuss the situation when multiple monitoring blocks are present in the address space. We assume for simplicity that the monitor blocks are pairwise disjoint and each has length m . Assume there are b monitor blocks, $b < k$, and $n = km$ is the total size of the circular address space identified with $\mathbf{Z}_n$ as before.

The Attacker can still start with a Round Robin. Randomly pick a cell $j_0 \in \mathbf{Z}_n$ to start. If the current query cell is j , then query it until receiving an answer $b=1$, or have queried it ℓ times. Then replace j by $j+m$. We repeat this until $j = j_0$ again. At this point, we will have queried $k = n/m$ locations. At each such location j we record the final bit b_j . Notice that these bits are all correct answers. We will show that, in the presence of multiple monitoring blocks, this Round Robin strategy followed by a certain ‘one-sided binary search’ (we denote it by OSBS) can achieve essentially as good an upper bound as when there is only one block of length m , regardless of Defender’s strategy. This justifies our consideration of *only* this Round Robin strategy for A . At the same time, a suitable Defender’s strategy can achieve essentially the same quantity as a lower bound for the Defender, and therefore we have characterized the game.

Consider those bits $b_j = 1$. These indicate the presence of the monitoring blocks. We observe that no two bits can refer to the same block and each block must make its presence known through one of these bits. The bits $b_j = 1$ partition the k queried locations into contiguous runs of 1’s separated by 0’s in a circular array of k bits. We will concentrate on one such run of 1’s. There is no interference between different runs

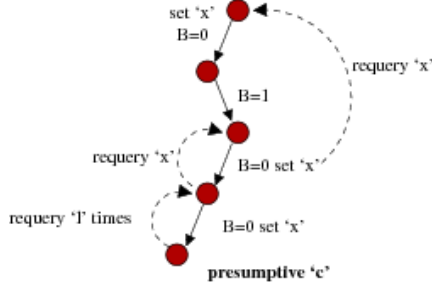


Fig. 1. Example OSBS traversal

of blocks indicated by the corresponding runs of 1's. Now consider one particular run of 1's. Without loss of generality suppose the run is $b_{rm} \dots b_m b_0 = 1 \dots 11$ and 0 is the right-most location where we have the bit $b_0 = 1$. This bit indicates the presence of a block, B^c , where $c \in \{0, 1, \dots, m-1\}$. OSBS will try to determine the location of c . After that, it can remove this (right-most) block B^c (in the run). If the run has more blocks left, we repeat OSBS on the new right-most block $B^{c'}$, with the slight modification that the right-most location for c' is $c - m$, i.e., $c' \in \{-m, \dots, c - m\}$. As $c \leq m-1$, the range is $c+1 \leq m$. Therefore the bound we prove for OSBS on B^c applies to every block in the run. Now consider OSBS on B^c . If D does not commit any lies during this search, then one can perform an ordinary binary search, i.e., if $m \geq 2$, our first query is at $i = \lfloor m/2 \rfloor$. The answer may be 0 or 1, indicating $c < i$ or $c \geq i$, and we continue until finding c . It is well known that this takes $\lceil \log_2 m \rceil$ steps.

Now we have to consider the complication that D can lie a total of $< \ell$ times. However, it can lie only in one direction, namely, if the query i is such that $c < i$, the answer must be $b = 0$. Only when $c \geq i$, does the Defender D have the option of lying with $b = 0$, while the true answer is $b = 1$.

OSBS algorithm: In view of this, we carry out our OSBS as follows. We select our next new query point just as in ordinary binary search, as if there were no lies. Record the answer bits as $\beta_1 \beta_2 \dots$, where for each $\beta_i = 0$ or 1 we branch left or right, respectively. If any $\beta_i = 0$ at a cell x , however, we modify ordinary binary search as follows. We will come back to do a 'repeated querying' at this location x , one query per each step, as we descend in the binary search to its left, until either:

- (a) we reach bottom (finding a presumptive c). If so, confirm $\beta_i = 0$ by querying ℓ times, else go to (b).
- (b) we discover $\beta_i = 0$ at x was a lie. If so, abandon work to the left of x , resume at x branching to the right.
- (c) have made ℓ queries at x , confirming $\beta_i = 0$ is true. If so, resume OSBS with no more queries at x .
- (d) got a new bit $\beta_j = 0$ at a cell y , further down in the binary search (i.e., $j > i$). If so, replace y with x and proceed recursively.

Note that in case (b), it is possible that, due to recursion, there is a ‘higher’ x' in the binary search tree where the corresponding bit $\beta_{x'} = 0$, and the ‘repeated querying’ at x' was suspended because of x by case (d) for x' . At this point we resume the ‘repeated querying’ at x' . The search ends by finding c and the confirmation that the smallest cell to its right for which the bit $\beta_i = 0$ is not a lie (by having made a total of ℓ queries there). If all $\beta_i = 1$, then we must end in $c = m - 1$. This must be correct and there were no lies committed. In Figure 1, we illustrate OSBS traversal with an example.

We claim that we make at most $O(\ell + \log m)$ queries in total. It is clear that this bound holds if no lies are committed. The $O(\log m)$ pays for the ‘binary search’ and $O(\ell)$ pays for the final confirmation. Suppose at some step during OSBS a lie was committed at x . We observe the following: in the descent in the binary search to the left of x , all subsequent answers with a $\beta = 0$ are lies. In this case, every step of the ‘repeated querying’ at x , as well as recursively all ‘repeated querying’ at locations to the left of x with a $\beta = 0$ in OSBS is charged to a total quota of $< \ell$ lies. Each of up to ℓ lies pays for $O(1)$ queries. This pays for the ordinary binary search queries at new locations descending from x as well. We also note that in case (c) above, a confirmation at x for a $\beta_i = 0$ also confirms all ‘higher’ x' in the binary search tree where the bit $\beta_{x'} = 0$.

In short, OSBS makes $O(\ell + \log m)$ queries in total. Corresponding to an ordinary binary search, it does the ‘repeated querying’, which costs at most two queries per each ordinary binary search step. In addition, when OSBS discovers a lie at x , it abandons the portion of the work done after x . But that amount of work is proportional to the number of queries made at a location with a lie, and therefore costs $O(\ell)$. Every lie $\beta = 0$ is eventually discovered. The total amount of work consists of (1) at most double the work in ordinary binary search, and (2) the abandoned work (due to the discovery of lies) which is at most $O(\ell)$. Finally it costs $O(\ell)$ to confirm the answer. This proves the upper bound $O(\ell + \log m)$.

It is also clear that $O(\ell + \log m)$ is optimal up to a constant factor. The Defender can use $O(\ell)$ to delay, and information theoretically it takes $O(\log m)$ to find the $c \in \{0, \dots, m - 1\}$. Overall, for b blocks, as $b < k$, we get a simultaneous upper and lower bound $\Theta(k\ell + b(\ell + \log m)) = \Theta(k\ell + b \log m)$. The lower bound means that the Defender can achieve this on the average. If $k\ell \geq b \log m$, then clearly it requires this much with DD (even with only one block). If $k\ell < b \log m$, we can imagine that all b blocks are separated, then information theoretically we have the bound $\Omega(b \log m) = \Omega(k\ell + b \log m)$.

5 Summary and Conclusion

In the perennial struggle against network intruders and malicious attacks, safeguarding honeynet monitors is becoming an urgent problem. This paper abstracts the problem in a game theoretic framework, and analyzes optimal strategies for both the Attacker and Defender. To achieve provable results and mathematical elegance, it is necessary to abstract away many systems details. But these abstractions aim to capture the most salient features of the network reality, and to achieve a reasonable balance of system relevance and theoretical tractability. As far as we know, our paper is the first to provide

a theoretical foundation for honeynet defense. It has also proven useful in guiding the development of Kaleidoscope, an experimental middlebox for safeguarding honeynet monitors. Our experience with Kaleidoscope also reveals a number of system issues and variants that can be further analyzed in a game theoretic setting.

Acknowledgements

This work was supported in part by the National Science Foundation (NSF) grants CCF-0830488, CCF-0511679, CNS-0716460, CNS-0831427 and CNS-0716612 and the U.S. Army Research Office (ARO) under the Cyber-TA research grant. Any opinions, findings, conclusions or other recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the NSF or U.S. ARO.

References

1. Bethencourt, J., Franklin, J., Vernon, M.: Mapping Internet Sensors with Probe Response Packets. In: Proceedings of USENIX Security Symposium (2005)
2. Cai, J.-Y., Yegneswaran, V., Alfeld, C., Barford, P.: Honeygames: A Game Theoretic Approach to Defending Network Monitors. In: University of Wisconsin, Technical Report #1577 (2006)
3. Cooke, E., Bailey, M., Mao, M., Watson, D., Jahanian, F., McPherson, D.: Toward Understanding Distributed Blackhole Placement. In: Proceedings of CCS Workshop on Rapid Malcode (WORM 2004) (October 2004)
4. German Honeynet Project. Tracking Botnets (2005), <http://www.honeynet.org/papers/bots>
5. Pang, R., Yegneswaran, V., Barford, P., Paxson, V., Peterson, L.: Characteristics of Internet Background Radiation. In: Proceedings of the ACM SIGCOMM Internet Measurement Conference (2004)
6. Provos, N.: A virtual honeypot framework. In: Proceedings of USENIX Security Symposium (2004)
7. Rajab, M.A., Monrose, F., Terzis, A.: Fast and evasive attacks: Highlighting the challenges ahead. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 206–225. Springer, Heidelberg (2006)
8. Shinoda, Y., Ikai, K., Itoh, M.: Vulnerabilities of Passive Internet Threat Monitors. In: Proceedings of USENIX Security Symposium (2005)
9. Staniford, S., Paxson, V., Weaver, N.: How to Own the Internet in Your Spare Time. In: Proceedings of the 11th USENIX Security Symposium (2002)
10. Ullrich, J.: Dshield (2005), <http://www.dshield.org>
11. Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker, G., Savage, S.: Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm. In: Proceedings of ACM SOSP 2005, Brighton, UK (October 2005)
12. Yegneswaran, V., Alfeld, C., Barford, P., Cai, J.-Y.: Camouflaging Honeynets. In: Proceedings of IEEE Global Internet Symposium (2007)
13. Yegneswaran, V., Barford, P., Plonka, D.: On the Design and Use of Internet Sinks for Network Abuse Monitoring. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 146–165. Springer, Heidelberg (2004)

On the Performances of Nash Equilibria in Isolation Games

Vittorio Bilò¹, Michele Flammini², Gianpiero Monaco², and Luca Moscardelli³

¹ Dipartimento di Matematica “Ennio De Giorgi” - Università del Salento
Provinciale Lecce-Arnesano P.O. Box 193, 73100 Lecce, Italy
`vittorio.bilo@unile.it`

² Dipartimento di Informatica - Università di L'Aquila
Via Vetoio, Coppito 67100 L'Aquila, Italy
`{flammini,gianpiero.monaco}@di.univaq.it`

³ Dipartimento di Scienze - Università di Chieti-Pescara
Viale Pindaro 42, 65127 Pescara, Italy
`moscardelli@sci.unich.it`

Abstract. We study the performances of Nash equilibria in isolation games, a class of competitive location games recently introduced in [14]. For all the cases in which the existence of Nash equilibria has been shown, we give tight or asymptotically tight bounds on the prices of anarchy and stability under the two classical social functions mostly investigated in the scientific literature, namely, the minimum utility per player and the sum of the players' utilities. Moreover, we prove that the convergence to Nash equilibria is not guaranteed in some of the not yet analyzed cases.

1 Introduction

Competitive location [6] is a multidisciplinary field of research and applications ranging from economy and geography to operations research, game theory and social sciences. It studies games in which players aim at choosing suitable locations or points in given metric spaces so as to maximize their utility or revenue. Depending on different parameters such as the underlying metric space, the number of players, the adopted equilibrium concept, the customers' behavior and so on, several scenarios arise. The foundations of competitive location date back to the beginning of the last century when Hotelling studied the so called “ice-cream vendor problem” [8]. With the recent flourishing of contributions on Algorithmic Game Theory, competitive location has attracted also the interest of computer scientists about the existence of equilibrium solutions, the complexity of their determination and the efficiency of these equilibria when compared with cooperative solutions optimizing a given social function measuring the overall welfare of the system.

In this paper we consider *isolation games* [14], a class of competitive location games in which the utility of a player is defined as a function of her distances from the other ones. For example, one can define the utility of a player as being equal to the distance from the nearest one (*nearest-neighbor isolation game*), or to the

sum of the distances from all the other players (*total-distance isolation game*), or to the distance from the ℓ -th nearest player (*ℓ -selection isolation game*). More generally, denoted as k the number of players, for any player i , each strategy profile S yields a vector $\mathbf{f}^i(S) \in \mathbb{R}_{\geq 0}^{k-1}$ such that the j -th component of $\mathbf{f}^i(S)$ is the distance between the location chosen by player i and the one chosen by her j -th nearest player. The utility of player i can thus be defined as any convex combination of the elements of $\mathbf{f}^i(S)$, that is, given a vector $\mathbf{w} \in \mathbb{R}_{\geq 0}^{k-1}$, the utility of player i is defined as the scalar product between $\mathbf{f}^i(S)$ and $\mathbf{w}$.

Isolation games find a natural application in data clustering [9] and geometric sampling [13]. Moreover, as pointed out in [14], they can be used to obtain a good approximation of the strategy a player should compute in another competitive location game, called Voronoi game [1,3,4,5,7,11], which is among the most studied competitive location games. Here the utility of a player is given by the total number of all points that are closer to her than to any other player (Voronoi area), where points which are equidistant to several players are split up evenly among the closest ones. Given a certain strategy profile, a player needs to compute the Voronoi area of any point in the space in order to play the game which can be very expensive in several situations. As an approximation, instead, each player may choose to simply maximize her nearest-neighbor distance or the total distance from any other players and this clearly gives rise to either a nearest-neighbor isolation game or a total-distance isolation game.

As another interesting field of application for isolation games, consider the following problem in non-cooperative wireless network design. We are given a set of users who have to select a spot where to locate an antenna so as to transmit and receive a radio signal. When more than just one antenna is transmitting contemporaneously, interference between two or more signals may occur. More specifically, it may be the case that antenna i receive, at the same time, h different signals $r_1, \dots, r_h$ of which only r_1 is really destined to antenna i . According to the main standard models in wireless network design, signal r_1 can be correctly received by antenna i if and only if the ratio between the power at which r_1 is received by antenna i and the sum of the powers at which the undesired $h - 1$ signals are received by antenna i is greater than a certain threshold. If users are selfish players interested in minimizing the amount of interference their antenna will be subject to, they will decide to locate it as far as possible from the other ones thus giving rise to a particular isolation game.

Related Work. Isolation games were recently introduced in [14], where the authors give several results regarding existence of pure Nash equilibria [12] and the convergence of better and best response dynamics to any of such equilibria. In particular they prove that in any symmetric space the nearest-neighbor and the total-distance isolation games are potential games, which implies existence of Nash equilibria and convergence of any better response dynamics. In case of asymmetric spaces, however, for any non-null vector $\mathbf{w}$, there always exists a game admitting no Nash equilibria. Moreover, in asymmetric spaces deciding whether the nearest-neighbor or the total-distance isolation games possess a

Nash equilibrium is NP-complete. On symmetric spaces, for ℓ -selection isolation games they show that Nash equilibria always exist and that there always exists a better response dynamics converging to an equilibrium even though the game is not a potential one (hence, even best response dynamics may lead to cycles). For isolation games defined by $\mathbf{w} = (1, 1, 0, \dots, 0)$ on symmetric spaces, determining whether there exists a Nash equilibrium is NP-complete. Finally, some existential results for general isolation games on certain particular Euclidean spaces are given.

Our Contribution. We analyze the efficiency of pure Nash equilibria for all the classes of isolation games introduced in [14] for which existence of such equilibria is always guaranteed, namely, the ℓ -selection isolation games for any $1 \leq \ell < k$ and the total-distance isolation games. Following the leading approach in Algorithmic Game Theory’s literature, we use either the minimum utility per player (*MIN*) and the sum of the players’ utilities (*SUM*) as social functions measuring the overall welfare of a strategy profile and adopt the notions of price of anarchy [10] and price of stability [2] as a measure of the quality of pure Nash equilibria when compared with the strategy profiles maximizing the social functions. All our results, summarized in Tables 1 and 2, are tight or asymptotically tight. We then show that isolation games yielded by the weight vector $\mathbf{w} = (0, \dots, 0, 1, \dots, 1)$ are not potential games, that is, better response dynamics may not converge to Nash equilibria.

Table 1. Results for ℓ -selection isolation games

Social Function	Price of Stability	Price of Anarchy
<i>MIN</i>	1	2
<i>SUM</i>	∞	∞

Table 2. Results for total-distance isolation games

Social Function	Price of Stability	Price of Anarchy
<i>MIN</i>	$1 \div \frac{k+1}{k-1}$	$2 \div 2^{\frac{k+1}{k-1}}$
<i>SUM</i>	1	2

Paper Organization. Next section contains the necessary definitions and notation. Sections 3 and 4 cover the study of both the prices of anarchy and stability for ℓ -selection and total-distance isolation games, respectively. In Section 5 we show that the cases yielded by the weight vectors $\mathbf{w} = (0, \dots, 0, 1, \dots, 1)$ are not potential games, while in Section 6, we address open problems and further research. Due to space limitations, some proofs are omitted.

2 Definitions and Notation

For any $n \in \mathbb{N}$, let $[n] := \{1, \dots, n\}$. Given an n -tuple $A = (a_1, \dots, a_n)$, we write (A_{-i}, x) to denote the n -tuple obtained from A by replacing a_i with x . For the ease of notation, we write $x \in A$ when there exists an index $i \in [n]$ such that $a_i = x$.

A *metric space* is a pair (X, d) where X is a set of *points* or *locations* and $d : X \times X \mapsto \mathbb{R}_{\geq 0}$ is a *distance* or *metric function* such that for any $x, y, z \in X$ it holds (i) $d(x, y) \geq 0$, (ii) $d(x, y) = 0 \Leftrightarrow x = y$, (iii) $d(x, y) = d(y, x)$ (*symmetry*), and (iv) $d(x, y) \leq d(x, z) + d(z, y)$ (*triangular inequality*).

An instance $I = ((X, d), k)$ of an *isolation game* is defined by a metric space (X, d) and a set of k players $\{1, \dots, k\}$ aiming at selfishly maximizing their own utility. The *strategy set* of each player is given by the set X of all the locations and the set of strategy profiles is $\mathcal{S} := X^k$. A *strategy profile* $S \in \mathcal{S}$ is thus a k -tuple $S = (s_1, \dots, s_k)$ where, for any $i \in [k]$, s_i is the location chosen by player i in S .

Given a strategy profile S , for any player $i \in [k]$, define the *distance vector* of i in S as $\mathbf{b}^i(S) = (d(s_i, s_1), \dots, d(s_i, s_{i-1}), d(s_i, s_{i+1}), \dots, d(s_i, s_k))$ and the *ordered distance vector* of i in S as the vector $\mathbf{f}^i(S) \in \mathbb{R}_{\geq 0}^{k-1}$ obtained from $\mathbf{b}^i(S)$ by sorting its components in non-decreasing order. Roughly speaking, if player p is the j -th nearest player to i in S , the j -th component of $\mathbf{f}^i(S)$ is equal to the distance between s_i and s_p .

For any vector $\mathbf{w} \in \mathbb{R}_{\geq 0}^{k-1}$, called *weight vector*, the utility $u_i(S)$ of player i in S is given by the scalar product between $\mathbf{f}^i(S)$ and $\mathbf{w}$, that is, $u_i(S) = \mathbf{f}^i(S) \cdot \mathbf{w}$. Informally speaking, w_ℓ denotes how much the distance from the ℓ -th furthest player influences the utility of any player. For instance, for the weight vector $\mathbf{w} = (1, 0, \dots, 0)$, the utility of a player is simply given by the distance from the nearest one.

In this paper, we consider the following weight vectors:

- **ℓ -selection vector** $(0, \dots, 0, 1, 0, \dots, 0)$, in which all components are equal to zero except for the ℓ -th one which is set to one. Therefore, the utility of a player is given by the distance from the ℓ -th nearest one. We call ℓ -selection isolation games, the games yielded by an ℓ -selection vector;
- **sum vector** $(1, \dots, 1)$, for which the utility of a player is given by the sum of the distances from all the other ones. We call total-distance isolation games, the games yielded by the sum vector;
- **ℓ -suffix vector** $(0, \dots, 0, 1, \dots, 1)$, in which the last ℓ components are equal to one and the other ones are set to zero. Hence, the utility of a player is given by the sum of the distances from the ℓ furthest ones. We call ℓ -suffix isolation games, the games yielded by an ℓ -suffix vector.

Given a strategy profile S , player i can perform an *improving move* in S if and only if there exists a location $x \in X$ such that $u_i((S_{-i}, x)) > u_i(S)$. A strategy profile is a *Nash equilibrium* if and only if no player can perform an improving move in it.

Given a *social function* $SF : \mathcal{S} \mapsto \mathbb{R}_{\geq 0}$, for each instance I , let $OPT_I = \max_{S \in \mathcal{S}} \{SF(S)\}$ be the *social optimum* of I . Denoted as $\mathcal{N}_I$ the set of Nash equilibria of I , the *price of anarchy* of I (denoted as PoA_I) is the worst case ratio between the social optimum and the social value of a Nash equilibrium, i.e., $PoA_I = \sup_{S \in \mathcal{N}_I} \frac{OPT_I}{SF(S)}$. Moreover, the *price of stability* of I (denoted as PoS_I) is the best case ratio between the social optimum and the social value of a Nash equilibrium, i.e., $PoS_I = \inf_{S \in \mathcal{N}_I} \frac{OPT_I}{SF(S)}$. Let $\mathcal{I}_{\mathcal{G}}$ be the set of all instances of a given class of games $\mathcal{G}$. The prices of anarchy and stability of $\mathcal{G}$ (denoted as $PoA_{\mathcal{G}}$ and $PoS_{\mathcal{G}}$, respectively) are defined as $PoA_{\mathcal{G}} = \sup_{I \in \mathcal{I}_{\mathcal{G}}} PoA_I$ and $PoS_{\mathcal{G}} = \sup_{I \in \mathcal{I}_{\mathcal{G}}} PoS_I$.

We study the prices of anarchy and stability of ℓ -selection and total-distance isolation games under the two standard social functions adopted in the literature, namely, the minimum utility per player $MIN(S) = \min_{i \in [k]} \{u_i(S)\}$ and the sum of the utilities of all the players $SUM(S) = \sum_{i \in [k]} u_i(S)$.

3 ℓ -Selection Isolation Games

As proved in [14], for these games the existence of Nash equilibria is guaranteed for any value of $\ell \in [k-1]$, therefore, we study the performances of such equilibria by tightly bounding the prices of anarchy and stability for both social functions MIN (Section 3.1) and SUM (Section 3.2).

3.1 The Social Function MIN

In the next theorem, we show a tight bound on the price of anarchy of ℓ -selection isolation games for any possible value of ℓ .

Theorem 1. *For any $\ell \in [k-1]$, the price of anarchy of ℓ -selection isolation games under the social function MIN is 2.*

Proof. We first provide an instance $I = ((X, d), k)$ for which there exists a Nash equilibrium of social value 1 in any possible ℓ -selection isolation game, while $OPT_I \geq 2$, thus establishing a lower bound of 2 on the price of anarchy.

Let $k = 2$ and (X, d) be a metric space such that $X = \{x_1, x_2, x_3, x_4\}$ and d is defined as shown in Table 3. Since ℓ cannot exceed $k-1$, we only have to consider nearest-neighbor isolation games to establish the lower bound.

Table 3. The distance function of the metric space used in the proof of Theorem 1

d	x_1	x_2	x_3	x_4
x_1	0	1	1	1
x_2	1	0	1	1
x_3	1	1	0	2
x_4	1	1	2	0

On the one hand, it is easy to check that in the nearest-neighbor isolation game played on instance I , the strategy profile $S = (x_1, x_2)$ is a Nash equilibrium in which each player has utility 1, and thus $MIN(S) = 1$; on the other hand, in the strategy profile $S^* = (x_3, x_4)$ each player has utility 2, and therefore $OPT_I \geq MIN(S^*) = 2$.

In order to complete the proof, it remains to show that for any instance of an ℓ -selection isolation game the price of anarchy is at most 2.

Consider a generic instance of an ℓ -selection isolation game, and let $S^* = (s_1^*, \dots, s_k^*)$ and $S = (s_1, \dots, s_k)$ be an optimal solution and a Nash equilibrium for it, respectively. We want to prove that $MIN(S^*) \leq 2MIN(S)$.

For every player $i \in [k]$, let $G(i) \subseteq [k]$, $|G(i)| = \ell$, be the set of the ℓ players closest to location s_i^* in S , breaking ties arbitrarily. Let $j \in [k]$ be a player having utility $u_j(S) = MIN(S)$; we have that for any $i \in [k]$, $d(s_i^*, s_x) \leq MIN(S)$ for every $x \in G(i)$ (we call such fact *main property*). In fact, assume by contradiction that there exist $i \in [k]$ and $x \in G(i)$ such that $d(s_i^*, s_x) > MIN(S)$; then, player j can improve her utility by migrating to the location s_i^* : a contradiction, because S is a Nash equilibrium.

For each $i \in [k]$, let $h(i)$ be the number of sets $G(j)$, $j \in [k]$, such that $i \in G(j)$; the proof is now divided into two cases.

- If there exists $i \in [k]$ such that $h(i) \geq \ell + 1$, by the *main property* there must be at least $\ell + 1$ players in S^* having distance at most $MIN(S)$ from location s_i . Thus, by the *symmetry* and *triangular inequality* properties of d , it follows that all such players have the ℓ -th furthest player at distance at most $2MIN(S)$. Therefore, $MIN(S^*) \leq 2MIN(S)$.
- If for all players $i \in [k]$, it holds $h(i) \leq \ell$, since $\sum_{i=1}^k h(i) = k\ell$, we have that $h(i) = \ell$ for all players $i \in [k]$. Let $i \in [k]$ be a player such that $j \in G(i)$, then there must exist a player $j' \notin G(i)$ such that $d(s_i^*, s_{j'}) \leq MIN(S)$. In fact, assume by contradiction that for all $j'' \notin G(i)$, it holds $d(s_i^*, s_{j''}) > MIN(S)$; then, player j can improve her utility by migrating to the location s_i^* : a contradiction, because S is a Nash equilibrium. Thus, since $h(j') = \ell$ and there exists a player i such that $d(s_i^*, s_{j'}) \leq MIN(S)$ and $j' \notin G(i)$, by the *main property* there must be at least $\ell + 1$ players in S^* having distance at most $MIN(S)$ from location $s_{j'}$. The claim follows by the same arguments exploited in the previous case. $\square$

In order to prove that the price of stability is 1 for ℓ -selection isolation games with $\ell > 1$, we first have to show that such a result holds for the particular case of the nearest-neighbor isolation game, that is, when $\ell = 1$.

Theorem 2. *The price of stability of the nearest-neighbor isolation game under the social function MIN is 1.*

Proof. Given an instance of the nearest-neighbor isolation game, let S^* be a strategy profile attaining the social optimum.

Since, as shown in [14], the nearest-neighbor isolation game is a potential game, any sequence of better responses starting from S^* has to lead to a

Nash equilibrium. Let $S^* = S_0, S_1, \dots, S_h = \hat{S}$ be one of such sequences, that is, such that $\hat{S}$ is a Nash equilibrium and denote as i_α the player performing the improving move in S_α . If there exists $\alpha \in \{0, \dots, h-1\}$ such that $MIN(S_\alpha) > MIN(S_{\alpha+1})$, by the symmetry of the distance function, we have $u_{i_\alpha}(S_\alpha) > u_{i_\alpha}(S_{\alpha+1})$: a contradiction. Hence, for each $\alpha = 0, \dots, h-1$, it must hold $MIN(S_\alpha) \leq MIN(S_{\alpha+1})$. Therefore, $OPT = MIN(S^*) \leq MIN(\hat{S})$. $\square$

Now we can prove the bound on the price of stability for any ℓ -selection isolation game.

Theorem 3. *For any $1 < \ell < k$, the price of stability of ℓ -selection isolation games under the social function MIN is 1.*

3.2 The Social Function SUM

Theorem 4. *For any $\ell \in [k-1]$, the prices of anarchy and stability of ℓ -selection isolation games under the social function SUM are unbounded.*

4 Total-Distance Isolation Games

It has been proved in [14] that total-distance isolation games are potential games thus implying the existence of Nash equilibria and convergence to one such an equilibrium starting from any initial strategy profile. Therefore, in this section we study again the prices of anarchy and stability of Nash equilibria by giving tight bounds for the social function SUM (Section 4.1) and asymptotically tight bounds for the social function MIN (Section 4.2).

4.1 The Social Function SUM

In the next two theorems we show exact bounds for both the prices of anarchy and stability of total-distance isolation games under the social function SUM .

Theorem 5. *The price of stability of total-distance isolation games under the social function SUM is 1.*

Theorem 6. *The price of anarchy of total-distance isolation games under the social function SUM is 2.*

Proof. For any instance of the total-distance isolation game, let $S^* = (s_1^*, \dots, s_k^*)$ be a strategy profile attaining the social optimum and $S = (s_1, \dots, s_k)$ be a Nash equilibrium. We define the complete bipartite graph $K_{k,k} = (U \cup V, E)$ where $U = \{u_1, \dots, u_k\}$ and $V = \{v_1, \dots, v_k\}$. We associate the weight $d(s_i^*, s_j)$ with each $(u_i, v_j) \in E$.

Consider the partition of E into k matchings $M_1, \dots, M_k$ and let M^* be the minimum weight one, that is, such that $\sum_{(u,v) \in M^*} d(u, v) \leq \min_{i \in [k]} \left\{ \sum_{(u,v) \in M_i} d(u, v) \right\}$.

In order to prove the claim, we need to show that $SUM(S^*) = \sum_{u,u' \in U} d(u,u') \leq 2 \sum_{v,v' \in V} d(v,v') = SUM(S)$.

By applying a standard averaging argument, we have that

$$(k-1) \sum_{(u,v) \in M^*} d(u,v) \leq \sum_{(u,v) \in E \setminus M^*} d(u,v). \quad (1)$$

Moreover, since S is a Nash equilibrium, it must hold $\sum_{v' \in V} d(v,v') \geq \sum_{v \neq v' \in V} d(u,v')$ for any $v \in V$ and $u \in U$. For any bijection $m : U \rightarrow V$, by summing up the k different inequalities that can be obtained for any disjoint pair of vertices $(u, m(u))$, we get

$$\sum_{v,v' \in V} d(v,v') \geq \sum_{u \in U} \sum_{m(u) \neq v' \in V} d(u,v'). \quad (2)$$

Consider the bijection m^* induced by M^* . Since d satisfies the triangular inequality, for any pair of nodes $u, u' \in U$, we have that $d(u, u')$ cannot be more than $d(u, m^*(u)) + d(u', m^*(u))$.

By summing up over all possible pairs of nodes in U , we obtain

$$\begin{aligned} \sum_{u,u' \in U} d(u,u') &\leq (k-1) \sum_{u \in U} d(u, m^*(u)) + \sum_{(u,v \neq m^*(u))} d(u,v) \\ &= (k-1) \sum_{(u,v) \in M^*} d(u,v) + \sum_{(u,v) \in E \setminus M^*} d(u,v) \\ \{\text{Because of Equation 1}\} &\leq 2 \sum_{(u,v) \in E \setminus M^*} d(u,v) \\ &= 2 \sum_{u \in U} \sum_{m^*(u) \neq v \in V} d(u,v) \\ \{\text{Because of Equation 2}\} &\leq 2 \sum_{v,v' \in V} d(v,v'). \end{aligned}$$

□

4.2 The Social Function MIN

In order to prove that the price of anarchy is at most $2 \frac{k+1}{k-1}$ and the price of stability is at most $\frac{k+1}{k-1}$, we first show a general property relating the two social values of certain Nash equilibria for total-distance isolation games.

Lemma 1. *For any instance I of total-distance isolation games, let S^* and $\bar{S}^*$ be two strategy profiles attaining the social optimum for I under the social functions MIN and SUM respectively. For any Nash equilibrium S such that $SUM(S) \geq \frac{1}{\alpha} \cdot SUM(\bar{S}^*)$, it holds $\frac{MIN(S^*)}{MIN(S)} \leq \alpha \cdot \frac{k+1}{k-1}$.*

By exploiting Lemma 1, it is possible to obtain asymptotically tight bounds on the prices of anarchy and of stability of total-distance isolation games under the social function MIN .

Theorem 7. *The price of anarchy of total-distance isolation games under the social function MIN is between 2 and $2^{\frac{k+1}{k-1}}$.*

Proof. The lower bound of 2 can be derived by using the same instance described in the proof of Theorem 1.

In order to prove an upper bound of $2^{\frac{k+1}{k-1}}$ it is sufficient to apply Lemma 1 with $\alpha = 2$, since, by Theorem 6, the price of anarchy of total-distance isolation games under the social function SUM is 2. $\square$

Theorem 8. *The price of stability of total-distance isolation games under the social function MIN is between 1 and $\frac{k+1}{k-1}$.*

Proof. As in the previous theorem, the proof directly follows from Lemma 1 with $\alpha = 1$, since, by Theorem 5, the price of stability of total-distance isolation games under the social function SUM is 1. $\square$

5 ℓ -Suffix Isolation Games

In this section we consider the games induced by the ℓ -suffix vector $(0, \dots, 0, 1, \dots, 1)$ for any $2 \leq \ell \leq k - 2$. We recall that the utility of a player is given by the sum of the distances from the ℓ furthest ones. These games, representing a natural generalization of the total-distance ones, have not been considered in [14].

In the following theorem, we show that convergence to a Nash equilibrium is not guaranteed in this case.

Theorem 9. *The ℓ -suffix isolation games are not potential games.*

6 Conclusions and Open Problems

We have studied the efficiency of pure Nash equilibria in both ℓ -selection and total-distance isolation games, that is, in all the cases among the ones analyzed in [14] for which such equilibria have been proven to exist. We achieved tight bounds in all the cases, even if for the case of total-distance isolation games under the social function MIN they are asymptotically optimal in the sense that they get tight when the number of players goes to infinity. Getting matching lower and upper bounds for low values of k is an interesting left open issue worth to be investigated.

As a natural generalization of total-distance isolation games, one can consider ℓ -prefix and ℓ -suffix isolation games. For ℓ -prefix games, [14] have proven that Nash equilibria cannot exist even in the simple basic case yielded by the weight vector $(1, 1, 0, \dots, 0)$. For the left over ℓ -suffix games, we have shown that even the simple basic case yielded by the weight vector $(0, 1, 1)$ may not be a potential game, thus encompassing all the possible ℓ -suffix isolation games which can be defined on an instance with four players. It is left open, however, establishing whether Nash equilibria are guaranteed to exist.

Concerning other possible future research directions, studying games yielded by more general and complicated weight vectors seems to be a very challenging task. Moreover, restricting to particular metric spaces, such as Euclidean spaces, looks intriguing and promising also from an application point of view (see for instance interferences in wireless networks as mentioned in the introduction). Clearly, all the positive results, that is the existence of equilibria and the upper bounds on both the prices of anarchy and stability, carry over also to these cases. It would be interesting to determine which properties a metric space should satisfy in order for the price of anarchy of the ℓ -selection and the total-distance isolation games to get lower than 2.

References

1. Ahn, H.K., Cheng, S.W., Cheong, O., Golin, M.J., Oostrum, R.: Competitive facility location: the Voronoi game. *Theoretical Computer Science* 310(1-3), 457–467 (2004)
2. Anshelevich, E., Dasgupta, A., Tardos, E., Wexler, T.: Near-Optimal Network Design with Selfish Agents. In: *Proc. of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 511–520. ACM Press, New York (2003)
3. Cheong, O., Har-Peled, S., Linial, N., Matousek, J.: The one-round Voronoi game. *Discrete and Computational Geometry* 31, 125–138 (2004)
4. Dürr, C., Thang, N.K.: Nash equilibria in Voronoi games on graphs. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) *ESA 2007. LNCS*, vol. 4698, pp. 17–28. Springer, Heidelberg (2007)
5. Eaton, B.C., Lipsey, R.G.: The principle of minimum differentiation reconsidered: Some new developments in the theory of spatial competition. *Review of Economic Studies* 42(129), 27–49 (1975)
6. Eiselt, H.A., Laporte, G., Thisse, J.F.: Competitive location models: A framework and bibliography. *Transportation Science* 27(1), 44–54 (1993)
7. Fekete, S.P., Meijer, H.: The one-round Voronoi game replayed. *Computational Geometry: Theory and Applications* 30, 81–94 (2005)
8. Hotelling, H.: Stability in competition. *Computational Geometry: Theory and Applications* 39(153), 41–57 (1929)
9. Jain, A.K., Murty, M.N., Flynn, P.J.: Data Clustering: A Review. *ACM Computing Surveys* 31(3) (1999)
10. Koutsoupias, E., Papadimitriou, C.H.: Worst-Case Equilibria. In: Meinel, C., Tison, S. (eds.) *STACS 1999. LNCS*, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
11. Mavronicolas, M., Monien, B., Papadopoulou, V.G., Schoppmann, F.: Voronoi games on cycle graphs. In: Ochmański, E., Tyszkiewicz, J. (eds.) *MFCS 2008. LNCS*, vol. 5162, pp. 503–514. Springer, Heidelberg (2008)
12. Nash, J.: Equilibrium Points in n -Person Games. *Proc. of the National Academy of Sciences* 36, 48–49 (1950)
13. Teng, S.H.: Low Energy and Mutually Distant Sampling. *Journal of Algorithms* 30(1), 42–67 (1999)
14. Zhao, Y., Chen, W., Teng, S.H.: The Isolation Game: A Game of Distances. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) *ISAAC 2008. LNCS*, vol. 5369, pp. 148–158. Springer, Heidelberg (2008)

Limits to List Decoding Random Codes

Atri Rudra*

Department of Computer Science and Engineering,
University at Buffalo, SUNY, Buffalo, NY, 14620
`atri@buffalo.edu`

Abstract. It has been known since [Zyablov and Pinsker 1982] that a random q -ary code of rate $1 - H_q(\rho) - \varepsilon$ (where $0 < \rho < 1 - 1/q$, $\varepsilon > 0$ and $H_q(\cdot)$ is the q -ary entropy function) with high probability is a $(\rho, 1/\varepsilon)$ -list decodable code. (That is, every Hamming ball of radius at most ρn has at most $1/\varepsilon$ codewords in it.) In this paper we prove the “converse” result. In particular, we prove that for *every* $0 < \rho < 1 - 1/q$, a random code of rate $1 - H_q(\rho) - \varepsilon$, with high probability, is not a (ρ, L) -list decodable code for any $L \leq \frac{c}{\varepsilon}$, where c is a constant that depends only on ρ and q . We also prove a similar lower bound for random linear codes.

1 Introduction

One of the central questions in the theory of error-correcting codes (henceforth just codes) is to determine the optimal tradeoff between the amount of redundancy used in a code and the amount of errors it can tolerate during transmission over a noisy channel. The first result in this vein is the seminal work of Shannon that precisely determined this tradeoff for a class of stochastic channels [11]. In this paper, we will look at the harsher adversarial noise model pioneered by Hamming [7], where we model the channel as an adversary. That is, other than a bound on the total number of errors, the channel can arbitrarily corrupt the transmitted message.

Under the adversarial noise model, it is well known that for the same amount of redundancy, lesser number of errors can be corrected than stochastic noise models (by almost a factor of two). This result follows from a simple argument that exploits the requirement that one always has to recover the transmitted message from the received transmission. However, if one relaxes the strict constraint of uniquely outputting the transmitted message to allow a list of messages to be output (with the guarantee that the transmitted message is in the list), then it can be shown that the optimal tradeoff between the amount of redundancy and the amount of correctable adversarial errors coincides with the tradeoff for certain stochastic noise models. This relaxed notion of outputting a list of possible transmitted messages, called *list decoding*, was put forth by Elias [2] and Wozencraft [14] in the late 1950’s.

We point out that in the notion of list decoding, the size of the output list of messages is a crucial quantity. In particular, one can always “successfully” list

* Research supported in part by NSF CAREER Award 0844796.

decode by outputting the list of all possible messages, in which case the problem becomes trivial. Thus, the concept of list decoding is only interesting when the output list is constrained to be “small.” This paper deals with quantifying the “smallness” of the *list size*.

Before we state our results more precisely, we quickly set up some notation (see Section 2 for more details). A code introduces redundancy by mapping a message to a (longer) *codeword*. The redundancy of a code is measured by its *rate*, which is the ratio of the the number of information symbols in the message to that in the codeword. Thus, for a code with *encoding* function $E : \Sigma^k \rightarrow \Sigma^n$, the rate equals k/n . The *block length* of the code equals n , and Σ is its alphabet. A code with an alphabet size of $q = |\Sigma|$ is called a q -ary code. The goal in *decoding* is to find, given a noisy received word, the actual transmitted codeword. We will generally talk about the fraction of errors that can be successfully decoded from and denote it by ρ . A code is called (ρ, L) -list decodable if for every received word there are at most L codewords that differ from the received word in at most ρ fraction of positions.

Zyablov and Pinsker established the following optimal tradeoff between the rate and ρ for list decoding [15]. They showed that there exists q -ary $(\rho, 1/\varepsilon)$ -list decodable codes of rate $1 - H_q(\rho) - \varepsilon$ for any $\varepsilon > 0$ (where $H_q(x) = x \log_q(q - 1) - x \log_q x - (1 - x) \log_q(1 - x)$ is the q -ary entropy function). Further, they showed that any q -ary (ρ, L) -list decodable of rate $1 - H_q(\rho) + \varepsilon$ needs L to be exponentially large in the block length of the code. Thus, the quantity $1 - H_q(\rho)$ exactly is the optimal rate for list decoding from ρ fraction of errors (with small lists). This quantity also matches the “capacity” of the q -ary Symmetric channel.

However, the result of Zyablov and Pinsker does not pin-point the optimal value of L for any (ρ, L) -list decodable code with rate $1 - H_q(\rho) - \varepsilon$. In particular, their results do not seem to imply any *lower bound* on L for such codes. This leads to the following natural question (which was also raised in [6]):

Question 1. *Do q -ary (ρ, L) -list decodable codes of rate $1 - H_q(\rho) - \varepsilon$ need the list size L to be at least $\Omega(1/\varepsilon)$?*

We now digress briefly to talk about algorithmic aspects and its implications for list decoding. For most applications of list decoding, the combinatorial guarantee of good list decodability must be backed up with efficient polynomial time list decoding algorithms. Note that this imposes an a priori requirement that the list size needs to be at most some polynomial in the block length of the code. Good list decodable codes with efficient list decoding algorithms have found many applications in theoretical computer science in general and complexity theory in particular (see for example the survey by Sudan [12] and Guruswami’s thesis [4, Chap. 12]). Such codes also have potential applications in traditional coding theory applications domains such as communication (cf. [9, Chap. 1]). One interesting contrast in these applications are the regimes of ρ that they operate in. The applications in complexity theory require ρ to be very close to $1 - 1/q$, while in the communication setting, ρ being closer to zero is the more interesting setting. Thus, the entire spectrum of ρ merits study.

We now return to Question 1. For *binary* (ρ, L) -list decodable codes with rate $1 - H(\rho) - \varepsilon$ Blinovsky provides some partial answers [1]. In particular, for $\rho = 1/2 - \sqrt{\varepsilon}$, a tight bound on L of $\Omega(1/\varepsilon)$ is shown in [1]. For smaller ρ (in particular for constant ρ independent of ε), the result in [1] implies a lower bound of $\Omega(\log(1/\varepsilon))$ on L .¹ Thus, Blinovsky's result implies a positive resolution of Question 1 only for binary codes for $\rho = 1/2 - \sqrt{\varepsilon}$. This result was extended to q -ary codes by Guruswami and Vadhan [6]. In particular, they show that any q -ary $(1 - 1/q - \sqrt{\varepsilon}, L)$ -list decodable code with *any* constant rate needs $L = \Omega(1/\varepsilon)$. The result in [6] is proved by a different and simpler proof than the one used in [1].

Unfortunately, it is not clear how to strengthen the proof of Blinovsky to answer Question 1 in the affirmative for binary codes. Further, the proof of Guruswami and Vadhan crucially uses the fact that ρ is very close to $1 - 1/q$. Given this, answering Question 1 in its full generality seems to be tricky. However, if we scale down our ambitions, as a special case of Question 1 one can ask if the answer for random codes is positive (say with high probability). That is

Question 2. *Do random q -ary (ρ, L) -list decodable codes of rate $1 - H_q(\rho) - \varepsilon$ with high probability need the list size L to be at least $\Omega(1/\varepsilon)$?*

The above is a natural question given the result that random codes with high probability need the list size to be at most $1/\varepsilon$ [15]. In particular, Question 2 asks if the analysis of [15] is tight.

In this paper, we answer Question 2 affirmatively. Our main result states that a random q -ary (ρ, L) -list decodable code of rate $1 - H_q(\rho) - \varepsilon$ with high probability needs L to be $\Omega((1 - H_q(\rho))/\varepsilon)$, which is tight for any constant $\rho < 1 - 1/q$. In fact, our results also hold if we restrict our attention to random *linear* codes.² We remark that our results are somewhat incomparable with those of [1,6]. On the one hand, our results give tight lower bounds for the range of values of ρ and q for which the previous works do not work. On the other hand, [1,6] are more general as their lower bounds work for *arbitrary* codes.

Proof Overview. We now will briefly sketch the main ideas in our proof. First we review the proof of the “positive” result of [15], as it is the starting point for our proof. In particular, we will argue that for a random code of rate $1 - H_q(\rho) - 1/L$, the probability that some received word $\mathbf{y}$ has some $L+1$ set of codewords within a relative Hamming distance of ρ is exponentially small in the block length of the code. Note that this implies that with high probability the code is (ρ, L) -list decodable. Let k and n denote the dimension and block length of the code (note that $k/n = 1 - H_q(\rho) - 1/L$). To bound the probability of a “bad” event, the proof first shows that for a fixed set of $L+1$ messages and a received word $\mathbf{y}$, the probability that all the corresponding $L+1$ codewords lie within a Hamming ball

¹ These bounds are implicit in [1]. See [10] for the calculations that imply the lower bounds claimed here.

² The story for random linear codes is a bit different from that of general codes, as for random linear codes only an upper bound of $q^{O(1/\varepsilon)}$ on L is known.

of radius ρn centered around $\mathbf{y}$ is at most $q^{-n(L+1)(1-H_q(\rho))}$. Thus, the probability that some set of $L+1$ codewords lie within the Hamming ball is (by the union bound) at most $\binom{q^k}{L+1} \cdot q^{-n(L+1)(1-H_q(\rho))} \leq q^{-n(L+1)(1-H_q(\rho)-k/n)} \leq q^{-n(1+1/L)}$. Again by the union bound, the probability that the bad event happens for some received word is at most $q^{-n/L}$, as required.

Let us recast the calculation above in the following manner. Consider a random code of rate $1 - H_q(\rho) - \varepsilon$. Then the expected number of received words that have some L codewords in a Hamming ball of radius ρn around it is at most $q^{-nL(1-H_q(\rho)-k/n-1/L)}$. As $k/n = 1 - H_q(\rho) - \varepsilon$, if we pick $L = \lceil \frac{1}{2\varepsilon} \rceil$ then the expected number of received words with L codewords in a Hamming ball of radius ρn is $q^{\varepsilon n L}$. Now this is encouraging news if we want to prove a *lower bound* on L . Unfortunately, the bound on expectation above is an upper bound. However, if somehow the corresponding events were disjoint then we will be in good shape as for disjoint events the union bound is tight.

The main idea of this paper is to make the relevant events in the paragraph above disjoint. In particular, for now assume that $\rho < 1/2(1-1/q)$ and consider a code Y of constant rate with distance 2ρ (by the Gilbert-Varshamov bound [3,13] such codes exist). Let $\mathbf{y}_1 \neq \mathbf{y}_2$ be codewords in this code. Now the events that a fixed set of L codewords lie inside the Hamming balls of (relative) radius ρ around $\mathbf{y}_1$ and $\mathbf{y}_2$ are *disjoint*. By doing the calculations a bit carefully one can show that this implies that in expectation, exponentially many $\mathbf{y} \in Y$ have some L set of codewords within relative Hamming distance of ρ . Thus, for some code, there exists some received word for which the output list size needs to be at least L . To convert this into a high probability event, we bound the variance of these events (again the notion of disjointness discussed above helps) and then appeal to Chebyshev's inequality.

The drawback of the approach above is that one can only prove the required tight lower bound on the list size for $\rho < 1/2(1-1/q)$. To push up ρ close to $1 - 1/q$, we will need the following idea. Instead of carving the space of received words into (exponentially many) Hamming balls of relative radius 2ρ , we will carve the space into exponentially many disjoint clusters with the following properties. Every vector in a cluster is within a relative Hamming distance of ρ from the cluster center. Further, the size of every cluster is very close to the volume of a Hamming ball of relative radius ρ . It turns out that this approximation is good enough for the proof idea in the previous paragraph to go through. Such a carving with high probability can be obtained from a random code of rate close to $1 - H_q(\rho)$ (the cluster centers are the codewords in this code). Interestingly, the proof of this claim is implicit in Shannon's original work.

Organization. In Section 2 we will set up some preliminaries including the existence of the special kind of carving mentioned in the paragraph above. We prove our main result in Section 3 and conclude with some open questions in Section 4. Due to space limitations some proofs have been omitted, which can be found in the companion technical report [10].

2 Preliminaries

For an integer $m \geq 1$, we will use $[m]$ to denote the set $\{1, \dots, m\}$.

We will now quickly review the basic concepts from coding theory that will be needed for this work. A code of *dimension* k and *block length* n over an alphabet Σ is a subset of Σ^n of size $|\Sigma|^k$. By abuse of notation we will also think of a code C as a map from elements in Σ^k to their corresponding codeword in Σ^n . The *rate* of such a code equals k/n . Each vector in C is called a codeword. Throughout this paper, we will use $q \geq 2$ to denote the size of the alphabet of a code. We will denote by $\mathbb{F}_q$ the field with q elements. A code C over $\mathbb{F}_q$ is called a linear code if C is a linear subspace of $\mathbb{F}_q^n$. In this case the dimension of the code coincides with the dimension of C as a vector space over $\mathbb{F}_q$.

We will use boldface letters to denote vectors and $\mathbf{0}$ will denote the all zeros-vector. The Hamming distance between two vectors $\mathbf{u}, \mathbf{v} \in \Sigma^n$, denoted by $\Delta(\mathbf{u}, \mathbf{v})$ is the number of places they differ in. The volume of a Hamming ball of radius d centered at $\mathbf{u}$ is defined as follows: $\text{Vol}_q(\mathbf{u}, d) = |\{\mathbf{v} | \Delta(\mathbf{u}, \mathbf{v}) \leq d\}|$. We will use the following well known bound (cf. [8]):

$$q^{H_q(\rho)n - o(n)} \leq \text{Vol}_q(\mathbf{y}, \rho n) \leq q^{H_q(\rho)n}, \quad (1)$$

for every $\mathbf{y} \in [q]^n$ and $0 < \rho < 1 - 1/q$.

The (minimum) distance of a code C is the minimum Hamming distance between any two pairs of distinct codewords from C . The relative distance is the ratio of the distance to the block length.

We will need the following notion of a carving of a vector space.

Definition 1. Let $n \geq 1$ and $q \geq 2$ be integers and let $0 < \rho < 1 - 1/q$ and $\gamma \geq 0$ be reals. Then $\mathcal{P} = (\mathcal{H}, \mathcal{B})$ is a (ρ, γ) -carving of $[q]^n$ if the following hold:

- (a) $\mathcal{H} \subseteq [q]^n$ and $\mathcal{B} : \mathcal{H} \rightarrow 2^{[q]^n}$.
- (b) For every $\mathbf{x} \neq \mathbf{y} \in \mathcal{H}$, $\mathcal{B}(\mathbf{x}) \cap \mathcal{B}(\mathbf{y}) = \emptyset$.
- (c) For every $\mathbf{y} \in \mathcal{H}$ and $\mathbf{x} \in \mathcal{B}(\mathbf{y})$, $\Delta(\mathbf{y}, \mathbf{x}) \leq \rho n$.
- (d) For every $\mathbf{y} \in \mathcal{H}$, $|\mathcal{B}(\mathbf{y})| \geq (1 - q^{-\gamma n}) \text{Vol}_q(\mathbf{0}, \rho n)$.

The size of $\mathcal{P}$ is $|\mathcal{H}|$.

In our proof we will need a (ρ, γ) -carving of $[q]^n$ of size $q^{\Omega(n)}$.

2.1 Existence of (ρ, γ) -Carvings

To get a feel for these kinds of carvings, let us first consider the special case when $0 < \rho < 1/2(1 - 1/q)$ and $\gamma = \infty$. Let $\mathcal{P} = (\mathcal{H}, \mathcal{B})$ be a (ρ, γ) -carving of $[q]^n$. Note that since $\gamma = \infty$, then by definition, $\mathcal{B}$ maps each element in $\mathcal{H}$ to Hamming balls of radius ρn around them. Thus, if we pick $\mathcal{H}$ to be a q -ary code of distance $2\rho n + 2$, then $\mathcal{P}$ does satisfy the conditions of Definition 1. By the Gilbert-Varshamov bound, we know that there exists $\mathcal{H}$ with $|\mathcal{H}| \geq q^{(1 - H_q(2\rho) - \varepsilon)n}$ for any $\varepsilon > 0$. Unfortunately, the limitation of $\rho < 1/2(1 - 1/q)$ is unsatisfactory. Next we show that we can remove this constraint at the expense of having a smaller γ . The proof is omitted due to space considerations.

Lemma 1. *Let $q \geq 2$ be an integer and let $0 < \rho < 1 - 1/q$ and $\gamma > 0$ be reals. Then for large enough n , there exist a (ρ, γ) -carving $\mathcal{P} = (\mathcal{H}, \mathcal{B})$ of $[q]^n$ of size at least $q^{(1-H_q(\rho)-2\gamma)n}$.*

We remark that the proof of Lemma 1 follows from Shannon's proof of the capacity of the q -ary Symmetric Channel (with cross-over probability ρ) [11]. In particular, picking $\mathcal{H}$ to be a random code of rate slightly less than $1 - H_q(\rho)$ satisfies the required property with high probability. As a corollary, this implies that for random codes, for *most* error patterns, list decoding up to a radius of ρ will output at most *one* codeword. The connection to Shannon's proof has been made before (cf. [5,9]).

3 Lower Bound for General Random Codes

We start with the main technical result of the paper.

Lemma 2. *Let $q \geq 2$ be an integer and let $0 < \rho < 1 - 1/q$ and $\varepsilon > 0$ be reals. If $\mathcal{P} = (\mathcal{H}, \mathcal{B})$ is a (ρ, ε) -carving of $[q]^n$ of size $q^{\alpha n}$ for some $\alpha > 0$ then the following holds for every large enough integer n . A random code of rate $1 - H_q(\rho) - \varepsilon$, with high probability, has at least α/ε codewords in a Hamming ball of radius ρn centered at some vector in $\mathcal{H}$.*

Lemmas 1 and 2 imply our main result.

Theorem 1 (Main Result). *Let $q \geq 2$ be an integer and $0 < \rho < 1 - 1/q$ and $\varepsilon > 0$ be reals. Then a random code of rate $1 - H_q(\rho) - \varepsilon$, with high probability, is not (ρ, L) list-decodable for any $L \leq c_{q,\rho}/\varepsilon$, where $c_{q,\rho} > 0$ is a real number that depends only on q and ρ .*

In the rest of the section, we will prove Lemma 2.

Define $L = \lfloor \frac{\alpha}{3\varepsilon} \rfloor$ and $k = \lfloor (1 - H_q(\rho) - \varepsilon)n \rfloor$. Let $\mathcal{M}$ be the set of L tuples of distinct messages from $[q]^k$. Note that $|\mathcal{M}| = \binom{q^k}{L}$. Let C be a random q -ary code of dimension k and block length n (i.e. each message is assigned a random independent vector from $[q]^n$).

We now define a few indicator variables that will be needed in the proof. For any $\mathbf{m} = (\mathbf{m}_1, \dots, \mathbf{m}_L) \in \mathcal{M}$ and $\mathbf{y} \in [q]^n$, define the indicator variable $X_{\mathbf{m},\mathbf{y}}$ as follows³: $X_{\mathbf{m},\mathbf{y}} = 1$ if and only if $\{C(\mathbf{m}_1), \dots, C(\mathbf{m}_L)\} \subseteq \mathcal{B}(\mathbf{y})$.

Note that if $X_{\mathbf{m},\mathbf{y}} = 1$ then $\mathbf{m}$ and $\mathbf{y}$ form a “witness” to the fact that the code C needs to have an output list size of at least L . We also define a related indicator random variable: $Y_{\mathbf{m}} = 1$ if and only if $\sum_{\mathbf{y} \in \mathcal{H}} X_{\mathbf{m},\mathbf{y}} \geq 1$.

Finally define the following random variable: $Z = \sum_{\mathbf{m} \in \mathcal{M}} Y_{\mathbf{m}}$.

Note that to prove Lemma 2 it suffices to show that with high probability $Z \geq 1$. To this end, we will first bound the expectation and variance of Z and then invoke Chebyshev's inequality to obtain the required high probability guarantee.

³ All the indicator variables should also depend on C but we suppress the dependence to make the expressions simpler.

We begin by proving a lower bound on $\mathbf{E}[Z]$. As the codewords in C are chosen uniformly and independently at random from $[q]^n$, for every $\mathbf{m} \in \mathcal{M}$ and $\mathbf{y} \in \mathcal{H}$, we have: $\Pr[X_{\mathbf{m},\mathbf{y}} = 1] = \left(\frac{|\mathcal{B}(\mathbf{y})|}{q^n}\right)^L$. By property (d) in Definition 1, this implies that

$$(1 - q^{-\varepsilon n})^L \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n}\right)^L \leq \Pr[X_{\mathbf{m},\mathbf{y}} = 1] \leq \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n}\right)^L \quad (2)$$

By property (b) in Definition 1, it follows that for any $\mathbf{m} \in \mathcal{M}$ and $\mathbf{y}_1 \neq \mathbf{y}_2 \in \mathcal{H}$, the events $X_{\mathbf{m},\mathbf{y}_1} = 1$ and $X_{\mathbf{m},\mathbf{y}_2} = 1$ are *disjoint* events. This along with the lower bound in (2) implies the following for every $\mathbf{m} \in \mathcal{M}$:

$$\mathbf{E}[Y_{\mathbf{m}}] = \sum_{\mathbf{y} \in \mathcal{H}} \Pr[X_{\mathbf{m},\mathbf{y}} = 1] \geq |\mathcal{H}|(1 - 2Lq^{-\varepsilon n}) \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n}\right)^L, \quad (3)$$

where in the inequality we have used the fact that for large enough n , $(1 - q^{-\varepsilon n})^L \geq (1 - 2Lq^{-\varepsilon n})$. Using the upper bound in (2) in the above, we also get the following bound:

$$\mathbf{E}[Y_{\mathbf{m}}] \leq |\mathcal{H}| \cdot \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n}\right)^L \quad (4)$$

By linearity of expectation, we have

$$\mathbf{E}[Z] = \sum_{\mathbf{m} \in \mathcal{M}} \mathbf{E}[Y_{\mathbf{m}}] \geq |\mathcal{M}| \cdot |\mathcal{H}| \cdot (1 - 2Lq^{-\varepsilon n}) \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n}\right)^L \quad (5)$$

$$\geq \binom{q^k}{L} \cdot \frac{q^{\alpha n}}{2} \cdot \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n}\right)^L \quad (6)$$

$$\geq \frac{q^{kL+\alpha n}}{2L^L} \cdot q^{-nL(1-H_q(\rho))-o(n)} \quad (7)$$

$$\geq q^{nL(1-H_q(\rho)-\varepsilon+\frac{\alpha}{L}-1+H_q(\rho))-o(n)} \quad (8)$$

$$\geq q^{2\varepsilon nL-o(n)}. \quad (9)$$

In the above (5) follows from (3) while (6) follows from the fact that for large enough n , $2Lq^{-\varepsilon n} \leq 1/2$ (and plugging in the values of $|\mathcal{M}|$ and $|\mathcal{H}|$). (7) follows from the lower bound in (1) and the lower bound $\binom{a}{b} \geq (a/b)^b$. (8) and (9) follow from the choice of k and L (and by absorbing the “extra” terms like $2L^L$ into the $o(n)$ term). Using (4) in the above instead of (3) gives us the following bound

$$\mathbf{E}[Z] \leq q^{2\varepsilon nL}. \quad (10)$$

Next, we bound the variance of Z . As a first step in that direction, we will upper bound $\mathbf{E}[Z^2]$. By definition, we have $\mathbf{E}[Z^2] = \sum_{\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M}} \Pr[Y_{\mathbf{m}^1} = 1 \text{ and } Y_{\mathbf{m}^2} = 1]$. By abuse of notation, for every $\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M}$, define $\mathbf{m}^1 \cap \mathbf{m}^2$

to be the set of vectors from $[q]^k$ that occur in both the tuples $\mathbf{m}^1$ and $\mathbf{m}^2$. Similarly, $\mathbf{m}^1 \cup \mathbf{m}^2$ will denote the set of vectors from $[q]^k$ that occur in $\mathbf{m}^1$ or $\mathbf{m}^2$. With this notation in place, let us rewrite the summation above:

$$\mathbf{E}[Z^2] = \sum_{i=1}^L \sum_{\substack{\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M} \\ |\mathbf{m}^1 \cap \mathbf{m}^2| = i}} \Pr[Y_{\mathbf{m}^1} = 1 \bigwedge Y_{\mathbf{m}^2} = 1] + \sum_{\substack{\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M} \\ \mathbf{m}^1 \cap \mathbf{m}^2 = \emptyset}} \Pr[Y_{\mathbf{m}^1} = 1 \bigwedge Y_{\mathbf{m}^2} = 1] \quad (11)$$

We will bound the two summations in (11) using the following observation. Note that if $\mathbf{m}^1 \cap \mathbf{m}^2 \neq \emptyset$ then for every $\mathbf{y}_1 \neq \mathbf{y}_2 \in \mathcal{H}$, both $X_{\mathbf{m}^1, \mathbf{y}_1}$ and $X_{\mathbf{m}^2, \mathbf{y}_2}$ cannot be 1 *simultaneously* (this follows from the definition of $X_{(\cdot, \cdot)}$ and property (b) in Definition 1).

We will now bound the first summation in (11). Fix $1 \leq i \leq L$ and $\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M}$ such that $|\mathbf{m}^1 \cap \mathbf{m}^2| = i$. By the definition of the indicator variable $Y_{(\cdot)}$,

$$\begin{aligned} \Pr[Y_{\mathbf{m}^1} = 1 \bigwedge Y_{\mathbf{m}^2} = 1] &= \sum_{\mathbf{y}_1 \in \mathcal{H}} \sum_{\mathbf{y}_2 \in \mathcal{H}} \Pr[X_{\mathbf{m}^1, \mathbf{y}_1} = 1 \bigwedge X_{\mathbf{m}^2, \mathbf{y}_2} = 1] \\ &= \sum_{\mathbf{y} \in \mathcal{H}} \Pr[X_{\mathbf{m}^1, \mathbf{y}} = 1 \bigwedge X_{\mathbf{m}^2, \mathbf{y}} = 1] \end{aligned} \quad (12)$$

$$= \sum_{\mathbf{y} \in \mathcal{H}} \left(\frac{\text{Vol}_q(\mathbf{y}, \rho n)}{q^n} \right)^{2L-i} \quad (13)$$

$$= |\mathcal{H}| \cdot \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n} \right)^{2L-i} \quad (14)$$

In the above, (12) follows from the discussion in the paragraph above. (13) follows from the fact that every message in $\mathbf{m}^1 \cup \mathbf{m}^2$ is mapped to an independent random vector in $[q]^n$ by our choice of C (note that $|\mathbf{m}^1 \cup \mathbf{m}^2| = 2L - i$). Finally, (14) follows from the fact that the volume of a Hamming ball is translation invariant.

Now the number of tuples $\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M}$ such that $|\mathbf{m}^1 \cap \mathbf{m}^2| = i$ is upper bounded by $\binom{L}{i}^2 q^{k(2L-i)} \leq 2^{2L} q^{k(2L-i)}$. Thus, by (14) we get

$$\begin{aligned} \sum_{i=1}^L \sum_{\substack{\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M} \\ |\mathbf{m}^1 \cap \mathbf{m}^2| = i}} \Pr[Y_{\mathbf{m}^1} = 1 \bigwedge Y_{\mathbf{m}^2} = 1] &\leq 2^{2L} |\mathcal{H}| \sum_{i=1}^L (q^{k-n} \text{Vol}_q(\mathbf{0}, \rho n))^{2L-i} \\ &\leq 2^{2L} |\mathcal{H}| \sum_{i=1}^L q^{(2L-i)n(1-H_q(\rho)-\varepsilon-1+H_q(\rho))} \leq 2^{2L} |\mathcal{H}| \sum_{i=1}^L q^{-\varepsilon n L} \leq L 2^{2L} \cdot q^{2\varepsilon n L} \end{aligned} \quad (15)$$

In the above the second inequality follows from our choice of k and the upper bound in (1). The third inequality follows from the fact that $i \leq L$ while the final inequality follows from the size of $\mathcal{H}$ and the choice of L .

We now proceed to upper bound the second summation in (11). Fix $\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M}$ such that $\mathbf{m}^1 \cap \mathbf{m}^2 = \emptyset$. By the definition of $Y_{(\cdot)}$,

$$\begin{aligned} \Pr[Y_{\mathbf{m}^1} = 1 \text{ and } Y_{\mathbf{m}^2} = 1] &= \sum_{\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{H}} \Pr[X_{\mathbf{m}^1, \mathbf{y}_1} = 1 \text{ and } X_{\mathbf{m}^2, \mathbf{y}_2} = 1] \\ &= \sum_{\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{H}} \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n} \right)^{2L} = (|\mathcal{H}|)^2 \cdot \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n} \right)^{2L} \end{aligned} \quad (16)$$

In the above the second equality follows from the fact that the messages in $\mathbf{m}^1 \cup \mathbf{m}^2$ are assigned random independent codewords from $[q]^n$. Since the number of tuples $\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M}$ such that $\mathbf{m}^1 \cap \mathbf{m}^2 = \emptyset$ is upper bounded by $|\mathcal{M}|^2$, by (16) we have

$$\begin{aligned} \sum_{\substack{\mathbf{m}^1, \mathbf{m}^2 \in \mathcal{M} \\ \mathbf{m}^1 \cap \mathbf{m}^2 = \emptyset}} \Pr[Y_{\mathbf{m}^1} = 1 \text{ and } Y_{\mathbf{m}^2} = 1] &\leq (|\mathcal{M}| \cdot |\mathcal{H}|)^2 \left(\frac{\text{Vol}_q(\mathbf{0}, \rho n)}{q^n} \right)^{2L} \\ &\leq \left(\frac{\mathbf{E}[Z]}{1 - 2Lq^{-\varepsilon n}} \right)^2 \leq (1 + 8Lq^{-\varepsilon n}) \cdot (\mathbf{E}[Z])^2. \end{aligned} \quad (17)$$

In the above the second inequality follows from (5) while the last inequality is true for large enough n .

We are finally ready to bound the variance of Z :

$$\sigma^2[Z] = \mathbf{E}[Z^2] - (\mathbf{E}[Z])^2 \leq L2^{2L} \cdot q^{2\varepsilon n L} + 8Lq^{-\varepsilon n} (\mathbf{E}[Z])^2 \leq q^{4\varepsilon n L - \varepsilon n + o(n)} \quad (18)$$

In the above, the first inequality follows from (11), (15) and (17). The final inequality follows from (10) and by absorbing the multiplicative constants in the $o(n)$ term. Recall that we set out to prove that $\Pr[Z \geq 1]$ is large. Indeed,

$$\Pr[Z < 1] \leq \Pr[|Z - \mathbf{E}[Z]| > \mathbf{E}[Z]/2] \leq \frac{4\sigma^2[Z]}{(\mathbf{E}[Z])^2} \leq \frac{4q^{4\varepsilon n L - \varepsilon n + o(n)}}{q^{4\varepsilon n L - o(n)}} \leq q^{-\varepsilon n/2},$$

as desired. In the above the first inequality follows from the fact that for large enough n , $\mathbf{E}[Z] \geq 2$ (the latter fact follows from (9)). The second inequality is the Chebyschev's inequality. The third inequality follows from (18) and (9). The last inequality is true for large enough n . The proof of Lemma 2 is now complete.

3.1 Lower Bound for Random Linear Codes

The following result analogous to Theorem 1 holds for linear codes.

Theorem 2 (Linear Codes). *Let $q \geq 2$ be an integer and $0 < \rho < 1 - 1/q$ and $\varepsilon > 0$ be reals. Then a random linear code of rate $1 - H_q(\rho) - \varepsilon$, with high probability, is not (ρ, L) list-decodable for any $L \leq c_{q, \rho}/\varepsilon$, where $c_{q, \rho} > 0$ is a real number that depends only on q and ρ .*

The proof of Theorem 2 is very similar to that of Theorem 1 and is omitted.

4 Open Questions

In this work we proved that a random q -ary (ρ, L) list decodable code of rate $1 - H_q(\rho) - \varepsilon$ needs L to be at least $\Omega((1 - H_q(\rho))/\varepsilon)$ with high probability. It would be nice if we can prove a lower bound of the form $L \geq c_q/\varepsilon$, where c_q is an absolute constant that only depends on q . The obvious open question is to resolve Question 1. We conjecture that the answer should be positive.

Acknowledgments. Thanks to Bobby Kleinberg for asking whether lower bounds on list sizes for list decoding random codes are known. We thank Venkatesan Guruswami, Madhu Sudan and Santosh Vempala for helpful discussions on related topics.

References

1. Blinovskiy, V.M.: Bounds for codes in the case of list decoding of finite volume. *Problems of Information Transmission* 22(1), 7–19 (1986)
2. Elias, P.: List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics, MIT (1957)
3. Gilbert, E.N.: A comparison of signalling alphabets. *Bell System Technical Journal* 31, 504–522 (1952)
4. Guruswami, V.: *List Decoding of Error-Correcting Codes*. LNCS, vol. 3282. Springer, Heidelberg (2004)
5. Guruswami, V.: Algorithmic Results in List Decoding. *Foundations and Trends in Theoretical Computer Science (FnT-TCS)*, NOW publishers 2(2) (2007)
6. Guruswami, V., Vadhan, S.P.: A lower bound on list size for list decoding. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) *APPROX 2005 and RANDOM 2005*. LNCS, vol. 3624, pp. 318–329. Springer, Heidelberg (2005)
7. Hamming, R.W.: Error Detecting and Error Correcting Codes. *Bell System Technical Journal* 29, 147–160 (1950)
8. MacWilliams, F.J., Sloane, N.J.A.: *The Theory of Error-Correcting Codes*. Elsevier/North-Holland, Amsterdam (1981)
9. Rudra, A.: *List Decoding and Property Testing of Error Correcting Codes*. PhD thesis, University of Washington (2007)
10. Rudra, A.: Limits to list decoding random codes. *Electronic Colloquium on Computational Complexity (ECCC)* 16(013) (2009)
11. Shannon, C.E.: A mathematical theory of communication. *Bell System Technical Journal* 27, 379–423, 623–656 (1948)
12. Sudan, M.: List decoding: Algorithms and applications. *SIGACT News* 31, 16–27 (2000)
13. Varshamov, R.R.: Estimate of the number of signals in error correcting codes. *Doklady Akademii Nauk* 117, 739–741 (1957)
14. Wozencraft, J.M.: List Decoding. *Quarterly Progress Report*, Research Laboratory of Electronics, MIT 48, 90–95 (1958)
15. Zyablov, V.V., Pinsker, M.S.: List cascade decoding. *Problems of Information Transmission* 17(4), 29–34 (1981) (in Russian); 236–240 (1982) (in English)

Algorithm for Finding k -Vertex Out-trees and Its Application to k -Internal Out-branching Problem

Nathann Cohen¹, Fedor V. Fomin², Gregory Gutin³, Eun Jung Kim³, Saket Saurabh²,
and Anders Yeo³

¹ INRIA – Projet MASCOTTE

2004 route des Lucioles, BP 93 F-06902

Sophia Antipolis Cedex, France

nathann.cohen@sophia.inria.fr

² Department of Informatics, University of Bergen

POB 7803, 5020 Bergen, Norway

{fedor.fomin, saket.saurabh}@ii.uib.no

³ Department of Computer Science

Royal Holloway, University of London

Egham, Surrey TW20 0EX, UK

{gutin, eunjung, anders}@cs.rhul.ac.uk

Abstract. An *out-tree* T is an oriented tree with exactly one vertex of in-degree zero and a vertex x of T is called *internal* if its out-degree is positive. We design randomized and deterministic algorithms for deciding whether an input digraph contains a subgraph isomorphic to a given out-tree with k vertices. Both algorithms run in $O^*(5.704^k)$ time. We apply the deterministic algorithm to obtain an algorithm of runtime $O^*(c^k)$, where c is a constant, for deciding whether an input digraph contains a spanning out-tree with at least k internal vertices. This answers in affirmative a question of Gutin, Razgon and Kim (Proc. AAIM'08).

1 Introduction

An *out-tree* is an oriented tree with exactly one vertex of in-degree zero called the *root*. In the k -OUT-TREE problem, we are given as input a digraph D and an out-tree T on k vertices, and the question is to decide whether D contains a subgraph isomorphic to T . In a seminal paper, which introduced Color Coding, Alon, Yuster, and Zwick [1] gave randomized and deterministic fixed-parameter tractable (FPT) algorithms for the k -OUT-TREE problem, running in time $O(2^{O(k)}n)$. It is easy to see (see [6]), that the actual runtime of the randomized and deterministic algorithms presented in [1] is $O^*((4e)^k)^1$ and $O^*(c^k)$ respectively, where $c \geq 4e$.

The main results of [1], however, were a new algorithmic approach called Color Coding and a randomized $O^*((2e)^k)$ algorithm for deciding whether a digraph contains a path on k vertices (the k -PATH problem). Chen et al. [4] and Kneis et al. [9] developed a modification of Color Coding, Divide-and-Color, and using it designed a randomized

¹ In this paper we often use the notation $O^*(f(k))$ instead of $f(k)(kn)^{O(1)}$, that is, O^* hides not only constants, but also polynomial coefficients.

$O^*(4^k)$ -time algorithm for k -PATH. Divide-and-Color used by Chen et al. [4] and Kneis et al. [9] is ‘symmetric’, that is, both colors play similar roles and the probability of coloring a vertex with one of the colors is 0.5. In this paper, we develop an asymmetric version of Divide-and-Color, that is, the two colors play different roles and the probability of coloring a vertex with one of the colors depends on the color. As a result we obtain the fastest known randomized and deterministic algorithms for the k -OUT-TREE problem, with running time $O^*(5.704^k)$.

It is worth mentioning two recent results on k -PATH due to Koutis [10] and Williams [16] based on an algebraic approach. Koutis [10] obtained a randomized $O^*(2^{3k/2})$ -time algorithm for k -PATH. Building on the ideas of [10], Williams [16] obtained a randomized $O^*(2^k)$ -time algorithm for k -PATH. While the randomized algorithms based on Color Coding and Divide-and-Color are not difficult to derandomize, it is not the case for the algorithms of Koutis [10] and Williams [16]. Thus, it is not known whether there are deterministic algorithms for k -PATH with running time $O^*(2^{3k/2})$. Moreover, it is not clear whether the randomized algorithms of Koutis [10] and Williams [16] can be extended to solve k -OUT-TREE.

The study of fast algorithms for k -OUT-TREE is a problem interesting in its own right. However, we provide an important application of our deterministic algorithm. The vertices of an out-tree T of out-degree zero (non-zero) are *leaves* (*internal vertices*) of T . An *out-branching* of a digraph D is a spanning subgraph of D which is an out-tree. The MINIMUM LEAF problem is to find an out-branching with the minimum number of leaves in a given digraph D . This problem is of interest in database systems [7] and the Hamilton path problem is its special case. Thus, in particular, MINIMUM LEAF is NP-hard. In this paper we study the following parameterized version of MINIMUM LEAF: given a digraph D and a parameter k , decide whether D has an out-branching with at least k internal vertices. We call this the k -INT-OUT-BRANCHING problem. It was studied for symmetric digraphs (that is, undirected graphs) by Prieto and Sloper [14,15] and for all digraphs by Gutin et al. [8]. Gutin et al. [8] obtained an algorithm of running time $O^*(2^{O(k \log k)})$ for k -INT-OUT-BRANCHING and asked whether the problem admits an algorithm of running time $O^*(2^{O(k)})$. We remark that no such algorithm were known even for the case of symmetric digraphs [14,15]. In this paper, we obtain an $O^*(2^{O(k)})$ -time algorithm for k -INT-OUT-BRANCHING using our deterministic algorithm for k -OUT-TREE and an out-tree generation algorithm.

For a digraph D we use $V(D)$ to denote its vertex set. For a set X of vertices of a subgraph H of a digraph D , $N_H^+(X)$ and $N_H^-(X)$ denote the sets of out-neighbors and in-neighbors of vertices of X in H , respectively. Sometimes, when a set has a single element, we will not distinguish between the set and its element. In particular, when H is an out-tree and x is a vertex of H which is not its root, the unique in-neighbor of x is denoted by $N_H^-(x)$. For an out-tree T , $\text{Leaf}(T)$ denotes the set of leaves in T and $\text{Int}(T) = V(T) - \text{Leaf}(T)$ stands for the set of internal vertices of T .

2 New Algorithms for k -OUT-TREE

In this section, we give a new randomized algorithm for k -OUT-TREE that uses asymmetric version of Divide-and-Color and several other ideas. We give a short description

of time analysis and a proof of correctness, and omit all the proofs due to the space restrictions. The proofs can be found in a preprint of this paper [6]. Finally, we give a short discussion of its derandomization. We start with a well known result, see [5].

Lemma 1. *Let T be an undirected tree and let $w : V \rightarrow \mathbb{R}^+ \cup \{0\}$ be a weight function on its vertices. There exists a vertex $v \in V(T)$ such that the weight of every subtree T' of $T - v$ is at most $w(T)/2$, where $w(T) = \sum_{v \in V(T)} w(v)$.*

Consider a partition $n = n_1 + \dots + n_q$, where n and all n_i are non-negative integers and a bipartition (A, B) of the set $\{1, \dots, q\}$. Let $d(A, B) := \left| \sum_{i \in A} n_i - \sum_{i \in B} n_i \right|$. Given a set $Q = \{1, \dots, q\}$ with a non-negative integer weight n_i for each element $i \in Q$, we say that a bipartition (A, B) of Q is *greedily optimal* if $d(A, B)$ does not decrease by moving an element from one partite set to another. The following procedure describes how to obtain a greedily optimal bipartition in $O(q \log q)$ time. For simplicity we write $\sum_{i \in A} n_i$ as $n(A)$.

Algorithm 1. Bipartition $(Q, \{n_i : i \in Q\})$

- 1: Let $A := \emptyset, B := Q$.
 - 2: **while** $n(A) < n(B)$ and there is an element $i \in B$ with $0 < n_i < d(A, B)$ **do**
 - 3: Choose such an element $i \in B$ with a largest n_i .
 - 4: $A := A \cup \{i\}$ and $B := B - \{i\}$.
 - 5: **end while**
 - 6: Return (A, B) .
-

Lemma 2. *Let Q be a set of size q with a nonnegative integer weight n_i for each $i \in Q$. The algorithm **Bipartition** $(Q, \{n_i : i \in Q\})$ finds a greedily optimal bipartition $A \cup B = Q$ in time $O(q \log q)$.*

Now we describe the new randomized algorithm for k -OUT-TREE. Let D be a digraph and let T be an out-tree on k vertices. Let us fix a vertex $t \in V(T)$ and a vertex $w \in V(D)$. We call a copy of T in D a T -isomorphic tree. We say that a T -isomorphic tree T_D in D is a (t, w) -tree if $w \in V(T_D)$ plays the role of t .

In the following algorithm **find-tree**, we have several arguments other than the natural arguments T and D . Two arguments are vertices t and v of T , and the last argument is a pair consisting of $L \subseteq V(T)$ and $\{X_u : u \in L\}$, where $X_u \subset V(D)$ and X_u 's are pairwise disjoint. The argument t indicates that we want to return, at the end of the current procedure, the set of vertices X_t such that there is a (t, w) -tree for every $w \in X_t$. The fact that $X_t \neq \emptyset$ is used for two purposes: (a) to conclude that we have a T -isomorphic tree in D ; and (b) the information X_t is used to construct a larger tree using the current T -isomorphic tree, as a building block. Here, X_t is a kind of 'join'.

The arguments $L \subseteq V(T)$ and $\{X_u : u \in L\}$ form a set of information on the location in D of the vertices playing the role of $u \in L$ obtained in the way we obtained X_t by a recursive call of the algorithm. Let T_D be a T -isomorphic tree; if for every $u \in L$, T_D is a (v, w) -tree for some $w \in X_u$ and $V(T_D) \cap X_u = \{w\}$, we say that T_D *meets the restrictions on L* . The algorithm **find-tree** intends to find the set X_t of vertices such that for every $w \in X_t$, there is a (t, w) -tree which meets the restrictions on L .

The basic strategy is as follows. We choose a pair T_A and T_B of subtrees of T such that $V(T_A) \cup V(T_B) = V(T)$ and T_A and T_B share only one vertex, namely v^* . We call such v^* a *splitting vertex*. We call recursively two ‘**find-tree**’ procedures on subsets of $V(D)$ to ensure that the subtrees playing the role of T_A and T_B do not overlap. The first call (line 15) tries to find X_{v^*} and the second one (line 18), using the information X_{v^*} delivered by the first call, tries to find X_t . Here t is a vertex specified as an input for the algorithm **find-tree**. In the end, the current procedure will return X_t .

A splitting vertex can produce several subtrees, and hence there are many ways to divide them into two groups (T_A and T_B). To make the algorithm efficient, we try to obtain as ‘balanced’ a partition (T_A and T_B) as possible. The algorithm **tree-Bipartition** is used to produce an almost ‘balanced’ bipartition of the subtrees. We further introduce an argument in our algorithm, which allows us to analyze the time complexity of our algorithm more accurately. The argument v is a vertex which indicates whether there is a predetermined splitting vertex. If $v = \emptyset$, we do not have a predetermined splitting vertex and hence we find one in the current procedure. Otherwise, we use the vertex v as a splitting vertex.

Let r be the root of T . To decide whether D contains a copy of T , it suffices to run **find-tree**($T, D, \emptyset, r, \emptyset, \emptyset$).

Lemma 3. *During the performance of **find-tree**($T, D, \emptyset, r, \emptyset, \emptyset$), the sets X_u , $u \in L$ are pairwise disjoint.*

Lemma 4. *Consider the algorithm **tree-Bipartition** and let (WH, BL) be a bipartition of $\{1, \dots, q\}$ obtained at the end of the algorithm. Then the partition $U_w := \bigcup_{i \in WH} V(T_i) \cup \{v^*\}$ and $U_b := \bigcup_{i \in BL} V(T_i)$ of $V(T)$ has the the following property.*

- 1) If $v^* = t$, moving a component T_i from one partite set to the other does not decrease the difference $d(w(U_w), w(U_b))$.
- 2) If $v^* \neq t$, either exchanging v^* and the component T_l or moving a component T_i , $i \neq v^*, l$ from one partite set to the other does not decrease the difference $d(w(U_w), w(U_b))$.

Consider the following equation:

$$\alpha^2 - 3\alpha + 1 = 0 \quad (1)$$

Let $\alpha^* := (3 - \sqrt{5})/2$ be one of its roots. In line 10 of the algorithm **find-tree**, if $\alpha < \alpha^*$ we decide to pass the present splitting vertex v^* as a splitting vertex to the next recursive call which gets, as an argument, a subtree with greater weight. Lemma 5 justifies this execution. It claims that if $\alpha < \alpha^*$, then in the next recursive call with a subtree of weight $(1 - \alpha)w(T)$, we have a more balanced bipartition with v^* as a splitting vertex. Actually, the bipartition in the next step is good enough to compensate for the increase in the running time incurred by the biased ($\alpha < \alpha^*$) bipartition in the present step. We will show this in details later.

Lemma 5. *Suppose that v^* has been chosen to split T for the present call to **find-tree** such that the weight of every subtree of $T - v^*$ is at most $w(T)/2$ and that $w(T) \geq 5$. Let α be defined as in line 8 and assume that $\alpha < \alpha^*$. Let $\{U_1, U_2\} = \{U_w, U_b\}$ such*

Algorithm 2. $\text{find-tree}(T, D, v, t, L, \{X_u : u \in L\})$

```

1: if  $|V(T) \setminus L| \geq 2$  then
2:   for all  $u \in V(T)$ : Set  $w(u) := 0$  if  $u \in L$ ,  $w(u) := 1$  otherwise.
3:   if  $v = \emptyset$  then Find  $v^* \in V(T)$  such that the weight of every subtree  $T'$  of  $T - v^*$  is at
      most  $w(T)/2$  (see Lemma 1) else  $v^* := v$ 
4:    $(WH, BL) := \text{tree-Bipartition}(T, t, v^*, L)$ .
5:    $U_w := \bigcup_{i \in WH} V(T_i) \cup \{v^*\}$ ,  $U_b := \bigcup_{i \in BL} V(T_i)$ .
6:   for all  $u \in L \cap U_w$ : color all vertices of  $X_u$  in white.
7:   for all  $u \in L \cap (U_b \setminus \{v^*\})$ : color all vertices of  $X_u$  in black.
8:    $\alpha := \min\{w(U_w)/w(T), w(U_b)/w(T)\}$ .
9:   if  $\alpha^2 - 3\alpha + 1 \leq 0$  (i.e.,  $\alpha \geq (3 - \sqrt{5})/2$ , see (1) and the definition of  $\alpha^*$  afterwards)
      then  $v_w := v_b := \emptyset$ 
10:  else if  $w(U_w) < w(U_b)$  then  $v_w := \emptyset$ ,  $v_b := v^*$  else  $v_w := v^*$ ,  $v_b := \emptyset$ .
11:   $X_t := \emptyset$ .
12:  for  $i = 1$  to  $\left\lceil \frac{2.51}{\alpha \alpha^k (1-\alpha)^{(1-\alpha)k}} \right\rceil$  do
13:    Color the vertices of  $V(D) - \bigcup_{u \in L} X_u$  in white or black such that for each vertex the
      probability to be colored in white is  $\alpha$  if  $w(U_w) \leq w(U_b)$ , and  $1 - \alpha$  otherwise.
14:    Let  $V_w$  ( $V_b$ ) be the set of vertices of  $D$  colored in white (black).
15:     $S := \text{find-tree}(T[U_w], D[V_w], v_w, v^*, L \cap U_w, \{X_u : u \in L \cap U_w\})$ 
16:    if  $S \neq \emptyset$  then
17:       $X_{v^*} := S$ ,  $L := L \cup \{v^*\}$ .
18:       $S' := \text{find-tree}(T[U_b \cup \{v^*\}], D[V_b \cup S], v_b, t, (L \cap U_b), \{X_u : u \in (L \cap U_b)\})$ .
19:       $X_t := X_t \cup S'$ .
20:    end if
21:  end for
22:  Return  $X_t$ .
23: else  $\{|V(T) \setminus L| \leq 1\}$ 
24:   if  $\{z\} = V(T) \setminus L$  then  $X_z := V(D) - \bigcup_{u \in L} X_u$ ,  $L := L \cup \{z\}$ .
25:    $L^o := \{\text{all leaf vertices of } T\}$ .
26:   while  $L^o \neq L$  do
27:     Choose a vertex  $z \in L \setminus L^o$  s.t.  $N_T^+(z) \subseteq L^o$ .
28:      $X_z := X_z \cap \bigcap_{u \in N_T^+(z)} N^-(X_u)$ ;  $L^o := L^o \cup \{z\}$ .
29:   end while
30:   return  $X_t$ 
31: end if

```

that $w(U_2) \geq w(U_1)$ and let $\{T_1, T_2\} = \{T[U_w], T[U_b \cup \{v^*\}]\}$ such that $U_1 \subseteq V(T_1)$ and $U_2 \subseteq V(T_2)$. Let α' play the role of α in the recursive call using the tree T_2 . In this case the following holds: $\alpha' \geq (1 - 2\alpha)/(1 - \alpha) > \alpha^*$.

For the selection of the splitting vertex v^* we have two criteria in the algorithm **find-tree**: (i) ‘found’ criterion: the vertex is found so that the weight of every subtree T' of $T - v^*$ is at most $w(T)/2$. (ii) ‘taken-over’ criterion: the vertex is passed on to the present step as the argument v by the previous step of the algorithm. The following statement is an easy consequence of Lemma 5.

Corollary 1. *Suppose that $w(T) \geq 5$. If v^* is selected with ‘taken-over’ criterion, then $\alpha > \alpha^*$.*

Algorithm 3. $\text{tree-Bipartition}(T, t, v^*, L)$

```

1:  $T_1, \dots, T_q$  are the subtrees of  $T - v^*$ .  $Q := \{1, \dots, q\}$ .  $w(T_i) := |V(T_i) \setminus L|, \forall i \in Q$ .
2: if  $v^* = t$  then
3:    $(A, B) := \text{Bipartition}(Q, \{n_i := w(T_i) : i \in Q\})$ 
4:   if  $w(A) \leq w(B)$  then  $WH := A, BL := B$ . else  $WH := B, BL := A$ .
5: else if  $t \in V(T_i)$  and  $w(T_i) - w(v^*) \geq 0$  then
6:    $(A, B) := \text{Bipartition}(Q, \{n_i := w(T_i) : i \in Q \setminus \{l\}\} \cup \{n_l := w(T_l) - w(v^*)\})$ .
7:   if  $l \in B$  then  $WH := A, BL := B$ . else  $WH := B, BL := A$ .
8: else  $\{t \in V(T_i) \text{ and } w(T_i) - w(v^*) < 0\}$ 
9:    $(A, B) := \text{Bipartition}((Q \setminus \{l\}) \cup \{v^*\}, \{n_i := w(T_i) : i \in Q \setminus \{l\}\} \cup \{n_{v^*} := w(v^*)\})$ .

10: if  $v^* \in A$  then  $WH := A - \{v^*\}, BL := B \cup \{l\}$ . else  $WH := B - \{v^*\}, BL := A \cup \{l\}$ .
11: end if
12: return  $(WH, BL)$ .
```

Due to Corollary 1 the vertex v^* selected in line 3 of the algorithm **find-tree** functions properly as a splitting vertex. In other words, we have more than one subtree of $T - v^*$ in line 4 with positive weights.

Lemma 6. *If $w(T) \geq 2$, then for each of U_w and U_b found in line 5 of by **find-tree** we have $w(U_w) > 0$ and $w(U_b) > 0$.*

Lemma 7. *Given a digraph D , an out-tree T and a specified vertex $t \in V(T)$, consider the set X_t (in line 22) returned by the algorithm $\text{find-tree}(T, D, v, t, L, \{X_u : u \in L\})$. If $w \in X_t$ then D contains a (t, w) -tree that meets the restrictions on L . Conversely, if D contains a (t, w) -tree for a vertex $w \in V(D)$ that meets the restrictions on L , then X_t contains w with probability larger than $1 - 1/e > 0.6321$.*

The time complexity of the Algorithm **find-tree** is given in the following theorem.

Theorem 1. *Algorithm **find-tree** has running time $O(n^2 k^\rho C^k)$, where $w(T) = k$ and $|V(D)| = n$, and C and ρ are defined and bounded as follows:*

$$C = \left(\frac{1}{\alpha^* \alpha^* (1 - \alpha^*)^{1 - \alpha^*}} \right)^{\frac{1}{\alpha^*}}, \quad \rho = \frac{\ln(1/6)}{\ln(1 - \alpha^*)}, \quad \rho \leq 3.724, \text{ and } C < 5.704.$$

Derandomization of the algorithm **find-tree** can be carried out using the method presented by Chen et al. [4] and based on the construction of (n, k) -universal sets studied in [11] (for details, see [6]). As a result, we obtain the following:

Theorem 2. *There is an $O(n^2 C^k)$ time deterministic algorithm that solves the k -OUT-TREE problem, where $C \leq 5.704$.*

3 Algorithm for k -INT-OUT-BRANCHING

A k -internal out-tree is an out-tree with at least k internal vertices. We call a k -internal out-tree *minimal* if none of its proper subtrees is a k -internal out-tree, or *minimal k -tree*

in short. The ROOTED MINIMAL k -TREE problem is as follows: given a digraph D , a vertex u of D and a minimal k -tree T , where k is a parameter, decide whether D contains an out-tree rooted at u and isomorphic to T . Recall that k -INT-OUT-BRANCHING is the following problem: given a digraph D and a parameter k , decide whether D contains an out-branching with at least k internal vertices. Finally, the k -INT-OUT-TREE problem is stated as follows: given a digraph D and a parameter k , decide whether D contains an out-tree with at least k internal vertices.

Lemma 8. *Let T be a k -internal out-tree. Then T is minimal if and only if $|\text{Int}(T)| = k$ and every leaf $u \in \text{Leaf}(T)$ is the only child of its parent $N^-(u)$.*

Proof. Assume that T is minimal. It cannot have more than k internal vertices, because otherwise by removing any of its leaves, we obtain a subtree of T with at least k internal vertices. Thus $|\text{Int}(T)| = k$. If there are sibling leaves u and w , then removing one of them provides a subtree of T with $|\text{Int}(T)|$ internal vertices.

Now, assume that $|\text{Int}(T)| = k$ and every leaf $u \in \text{Leaf}(T)$ is the only child of its parent $N^-(u)$. Observe that every proper subtree T' of T must exclude at least one leaf of T . This implies that the parents of the excluded leaves become leaves of T' and hence $|\text{Int}(T')| < |\text{Int}(T)| = k$. Thus, T is a minimal k -tree. $\square$

In fact, Lemma 8 can be used to generate all non-isomorphic minimal k -trees. First, build an (arbitrary) out-tree T^0 with k vertices. Then extend T^0 by adding a vertex x' for each leaf $x \in \text{Leaf}(T^0)$ with an arc (x, x') . The resulting out-tree T' satisfies the properties of Lemma 8. By Lemma 8, we know that any minimal k -tree can be constructed in this way.

Generating Minimal k -Tree (GMT) Procedure

- a. Generate a k -vertex out-tree T^0 and a set $T' := T^0$.
- b. For each leaf $x \in \text{Leaf}(T')$, add a new vertex x' and an arc (x, x') to T' .

Due to the following observation, to solve k -INT-OUT-TREE for a digraph D , it suffices to solve instances of ROOTED MINIMAL k -TREE for each vertex $u \in V(D)$ as a root and for each minimal k -tree T .

Lemma 9. *Any k -internal out-tree rooted at r contains a minimal k -tree rooted at r as a subtree.*

Similarly, the next two lemmas show that to solve k -OUT-BRANCHING for a digraph D , it suffices to solve instances of ROOTED MINIMAL k -TREE for each vertex $u \in S$ as a root and each minimal k -tree T , where S is the unique strongly connected component of D without incoming arcs.

Lemma 10 ([2]). *A digraph D has an out-branching rooted at vertex $r \in V(D)$ if and only if D has a unique strongly connected component S of D without incoming arcs and $r \in S$. One can check whether D has a unique strongly connected component and find one, if it exists, in time $O(m + n)$, where n and m are the number of vertices and arcs in D , respectively.*

Lemma 11. *Suppose a given digraph D with n vertices and m arcs has an out-branching rooted at vertex r . Then any minimal k -tree rooted at r can be extended to a k -internal out-branching rooted at r in time $O(m + n)$.*

Proof. Let T be a k -internal out-tree rooted at r . If T is spanning, there is nothing to prove. Otherwise, choose $u \in V(D) \setminus V(T)$. Since there is an out-branching rooted at r , there is a directed path P from r to u . This implies that whenever $V(D) \setminus V(T) \neq \emptyset$, there is an arc (v, w) with $v \in V(T)$ and $w \in V(D) \setminus V(T)$. By adding the vertex w and the arc (v, w) to T , we obtain a k -internal out-tree and the number of vertices T spans is strictly increased by this operation. Using breadth-first search starting at some vertex of $V(T)$, we can extend T into a k -internal out-branching in $O(n + m)$ time. $\square$

Since k -INT-OUT-TREE and k -INT-OUT-BRANCHING can be solved in the same way, we only deal with the k -INT-OUT-BRANCHING problem. We will assume that our input digraph contains a unique strongly connected component S . Our algorithm called *IOBA* for solving k -INT-OUT-BRANCHING for a digraph D runs in two stages. In the first stage, we generate *all* minimal k -trees using the GMT procedure described above. At the second stage, for each $u \in S$ and each minimal k -tree T , we check whether D contains an out-tree rooted at u and isomorphic to T using our algorithm from the previous section. We return TRUE if and only if we succeed in finding an out-tree H of D rooted at $u \in S$ which is isomorphic to a minimal k -tree.

In the literature, mainly rooted (undirected) trees and not out-trees are studied. However, every rooted tree can be made an out-tree by orienting every edge away from the root and every out-tree can be made a rooted tree by disregarding all orientations. Thus, rooted trees and out-trees are equivalent and we can use results obtained for rooted trees for out-trees.

Otter [13] showed that the number of non-isomorphic out-trees on k vertices is $t_k = O^*(2.95^k)$. We can generate all non-isomorphic rooted trees on k vertices using the algorithm of Beyer and Hedetniemi [3] which runs in time $O(t_k)$. We know that GMT procedure generates all minimal k -trees and hence the first stage of IOBA can be completed in time $O^*(2.95^k)$. In the second stage of IOBA, we try to find a subgraph isomorphic to a minimal k -tree T in D , using our algorithm from the previous section. For each T , the out-tree subgraph isomorphism algorithm runs in time $O^*(5.704^k)$. Since the number of vertices in T is at most $2k - 1$, the overall running time of the algorithm is $O^*(2.95^k \cdot 5.704^{2k-1}) = O^*(96^k)$.

We can reduce the time complexity with a refined analysis of the algorithm. The major contribution to the large constant 96 in the above analysis comes from the running time of our algorithm from the previous section, for which we used a rough upper bound on the number of vertices in a minimal k -tree. Most of the minimal k -trees have less than $k - 1$ leaves, which implies that the upper bound $2k - 1$ on the order of a minimal k -tree is too large for the majority of the minimal k -trees. Let $T(k)$ be the running time of IOBA. Then we have

$$T(k) = O^* \left(\sum_{k+1 \leq k' \leq 2k-1} (\# \text{ minimal } k - \text{trees on } k' \text{ vertices}) \times (5.704^{k'}) \right) \quad (2)$$

A minimal k -tree T' on k' vertices has $k' - k$ leaves, and thus the out-tree T^0 from which T' is constructed has k vertices of which $k' - k$ are leaves. Hence the number of minimal k -trees on k' vertices is the same as the number of non-isomorphic out-trees on

k' vertices with $k' - k$ leaves. Here an interesting counting problem arises. Let $g(k, l)$ be the number of non-isomorphic out-trees on k vertices with l leaves. Find a tighter upper bound on $g(k, l)$. To our knowledge, such a function has not been studied yet. Leaving it as a challenging open question, here we give an upper bound on $g(k, l)$ and use it for an improved analysis of $T(k)$. In particular we are interested in the case when $l \geq k/2$.

Consider an out-tree T^0 on $k \geq 3$ vertices which has αk internal vertices and $(1 - \alpha)k$ leaves. We want to obtain an upper bound on the number of such non-isomorphic out-trees T^0 . Let T^c be the subtree of T^0 obtained after deleting all its leaves and suppose that T^c has βk leaves. Assume that $\alpha \leq 1/2$ and notice that αk and βk are integers. Clearly $\beta < \alpha$.

Each out-tree T^0 with $(1 - \alpha)k$ leaves can be obtained by appending $(1 - \alpha)k$ leaves to T^c so that each of the vertices in $\text{Leaf}(T^c)$ has at least one leaf appended to it. Imagine that we have $\beta k = |\text{Leaf}(T^c)|$ and $\alpha k - \beta k = |\text{Int}(T^c)|$ distinct boxes. Then what we are looking for is the number of ways to put $(1 - \alpha)k$ balls into the boxes so that each of the first βk boxes is nonempty. Again this is equivalent to putting $(1 - \alpha - \beta)k$ balls into αk distinct boxes. It is an easy exercise to see that this number equals $\binom{k - \beta k - 1}{\alpha k - 1}$.

Note that the above number does not give the exact value for the non-isomorphic out-trees on k vertices with $(1 - \alpha)k$ leaves. This is because we treat an out-tree T^c as a labeled one, which may lead to us to distinguishing two assignments of balls even though the two corresponding out-trees T^0 's are isomorphic to each other.

A minimal k -tree obtained from T^0 has $(1 - \alpha)k$ leaves and thus $(2 - \alpha)k$ vertices. With the upper bound $O^*(2.95^{\alpha k})$ on the number of T^c 's by [13], by (2) we have that $T(k)$ equals

$$\begin{aligned} & O^* \left(\sum_{\alpha \leq 1/2} \sum_{\beta < \alpha} 2.95^{\alpha k} \binom{k - \beta k - 1}{\alpha k - 1} (5.704)^{(2 - \alpha)k} + \sum_{\alpha > 1/2} 2.95^{\alpha k} (5.704)^{(2 - \alpha)k} \right) \\ &= O^* \left(\sum_{\alpha \leq 1/2} \sum_{\beta < \alpha} 2.95^{\alpha k} \binom{k}{\alpha k} (5.704)^{(2 - \alpha)k} \right) + O^* \left(2.95^k (5.704)^{3k/2} \right) \\ &= O^* \left(\sum_{\alpha \leq 1/2} \left(2.95^\alpha \frac{1}{\alpha^\alpha (1 - \alpha)^{1 - \alpha}} (5.704)^{(2 - \alpha)} \right)^k \right) + O^*(40.2^k). \end{aligned}$$

The term in the sum over $\alpha \leq 1/2$ above is maximized when $\alpha = \frac{2.95}{2.95 + 5.704}$, which yields $T(k) = O^*(49.4^k)$. Thus, we conclude with the following theorem.

Theorem 3. k -INT-OUT-BRANCHING is solvable in time $O^*(49.4^k)$.

4 Conclusion

In this paper we gave an asymmetric version of Divide-and-Color technique of Chen et al. [4] and Rossmanith [9]. Using this we obtained the fastest known parameterized

algorithm for the k -OUT-TREE problem. It is interesting to see if this technique can be used to obtain faster algorithms for other parameterized problems.

As a byproduct of our work, we obtained the first $O^*(2^{O(k)})$ time algorithm for k -INT-OUT-BRANCHING. Towards this, we used the classical result of Otter [13], that the number of non-isomorphic trees on k vertices is $O^*(2.95^k)$. An interesting combinatorial problem is to refine this bound for trees having $\lfloor \alpha k \rfloor$ leaves for some $\alpha < 1$.

Acknowledgements. The research of Cohen was partially funded by the project IST fet Aeolus. The research of Gutin, Kim and Yeo was supported in part by EPSRC.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. *Journal of the ACM* 42, 844–856 (1995)
2. Bang-Jensen, J., Gutin, G.: *Digraphs: Theory, Algorithms and Applications*, 2nd edn. Springer, London (2009)
3. Beyer, T., Hedetniemi, S.M.: Constant time generation of rooted trees. *SIAM J. Computing* 9, 706–712 (1980)
4. Chen, J., Lu, S., Sze, S.-H., Zhang, F.: Improved Algorithms for Path, Matching, and Packing Problems. In: *Proc. 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007)*, pp. 298–307 (2007)
5. Chung, F.R.K.: Separator theorems and their applications. In: Korte, B., Lovász, L., Prömel, H.J., Schrijver, A. (eds.) *Paths, Flows, and VLSI-Layout*, pp. 17–34. Springer, Berlin (1990)
6. Cohen, N., Fomin, F.V., Gutin, G., Kim, E.J., Saurabh, S., Yeo, A.: Algorithm for Finding k -Vertex Out-trees and its Application to k -Internal Out-branching Problem, Preprint arXiv:0903.0938 (March 2009)
7. Demers, A., Downing, A.: Minimum leaf spanning tree. US Patent no. 6,105,018 (August 2000)
8. Gutin, G., Razgon, I., Kim, E.J.: Minimum Leaf Out-Branching Problems. In: Fleischer, R., Xu, J. (eds.) *AAIM 2008. LNCS*, vol. 5034, pp. 235–246. Springer, Heidelberg (2008)
9. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Divide-and-color. In: Fomin, F.V. (ed.) *WG 2006. LNCS*, vol. 4271, pp. 58–67. Springer, Heidelberg (2006)
10. Koutis, I.: Faster algebraic algorithms for path and packing problems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part I. LNCS*, vol. 5125, pp. 575–586. Springer, Heidelberg (2008)
11. Naor, M., Schulman, L.J., Srinivasan, A.: Splitters and Near-Optimal Derandomization. In: *Proc. 17th Ann. Symp. Found. Comput. Sci.*, pp. 182–193 (1995)
12. Nilli, A.: Perfect hashing and probability. *Combinatorics Prob. Comput.* 3, 407–409 (1994)
13. Otter, R.: The Number of Trees. *Ann. Math.* 49, 583–599 (1948)
14. Prieto, E., Sloper, C.: Either/Or: Using Vertex Cover Structure in designing FPT-algorithms - The Case of k -Internal Spanning Tree. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) *WADS 2003. LNCS*, vol. 2748, pp. 474–483. Springer, Heidelberg (2003)
15. Prieto, E., Sloper, C.: Reducing To Independent Set Structure - The Case of k -Internal Spanning Tree. *Nordic Journal of Computing* 15, 308–318 (2005)
16. Williams, R.: Finding a path of length k in $O^*(2^k)$ time. *Inform. Proc. Letters.* 109(6), 315–318 (2009)

A $(4n - 4)$ -Bit Representation of a Rectangular Drawing or Floorplan

Toshihiko Takahashi¹, Ryo Fujimaki¹, and Youhei Inoue²

¹ Niigata University, Niigata 950-2181, Japan
takahasi@ie.niigata-u.ac.jp, fujimaki@magic.ie.niigata-u.ac.jp
² Renesas Technology Corp., Tokyo 100-0004, Japan
inoue.yohei@renesas.com

Abstract. A subdivision of a rectangle into rectangular faces with horizontal and vertical line segments is called a rectangular drawing or floorplan. Yamanaka and Nakano published a $(5n - 5)$ -bit representation of a rectangular drawing, where n is the number of inner rectangles. In this paper, a $(4n - 4)$ -bit representation of rectangular drawing is introduced. Moreover, this representation gives an alternative proof that the number of rectangles with n rectangles $R(n) \leq 13.5^{n-1}$.

1 Introduction

A *rectangular drawing* or *floorplan* is a subdivision of a rectangle with horizontal and vertical line segments such that no two line segments decussate. In the following, we use only the term “rectangular drawing” since a “floorplan” often means a placement of rectangles [1]. Two rectangular drawings are equivalent if (i) they have the same adjacent relations between the line segments and the rectangles and (ii) they have the same adjacent relations between the rectangles. The number of the inner rectangles, simply rectangles, of a rectangular drawing is denoted by n . $R(n)$ denotes the number of different rectangular drawings with n rectangles. For example, $R(4) = 24$. Fig.1 shows the 24 different rectangular drawings.

Several representations of rectangular drawings have been published: For example, H-Sequence [2], EQ-Sequence [3], FT-Squeeze [4], and so on. These representations are motivated by the VLSI physical design, where simple and tractable representations are required by layout algorithms.

Yamanaka and Nakano published a $(5n - 5)$ -bit representation of a rectangular drawing, which is the shortest representation ever published [5]. This result is also interesting from the viewpoint of enumerative combinatorics: There has not been a closed formula for $R(n)$, however, the result gives an upper bound, that is, $R(n) \leq 2^{5n-5} = 32^{n-1}$. The best result on upper bounds of $R(n)$ is that $R(n) \leq 13.5^{n-1}$ [6].

In this paper, a $(4n - 4)$ -bit representation of rectangular drawings is introduced. Moreover, an alternative proof that $R(n) \leq 13.5^{n-1}$ is obtained in consequence.

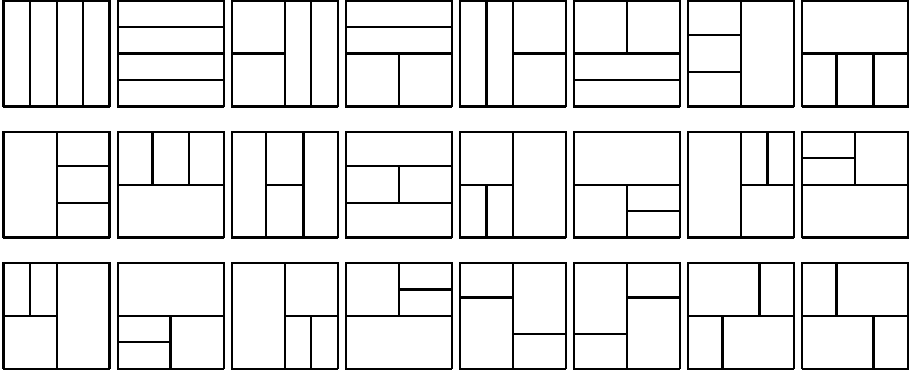


Fig. 1. 24 rectangular drawings for $n = 4$

This paper is organized as follows: Section 2 explains some notions first introduced in [7]. Section 3 gives a $(4n - 4)$ -bit representation of a rectangle drawing. In section 4, $R(n) \leq 13.5^{n-1}$ is proved by the representation.

2 Staircase and Deletable Rectangle

This section gives the notions of staircase and deletable rectangles, which were introduced to develop an algorithm computing $R(n)$ in polynomial time [7].

2.1 Staircase

A *staircase* is a configuration in the first quadrant of xy -plane such that

- the boarder consists of two line segments on x -axis and y -axis and a monotonic decreasing rectilinear path i.e., polygonal line of horizontal and vertical line segments (Fig.2).
- the interior is subdivided into rectangles with horizontal and vertical line segments, and
- no two line segments decussate.

Horizontal line segments of stairs, except on the x -axis, are called *steps*. The number of steps in a staircase is denoted by m . The i -th step from the bottom is simply called i -th step.

The number of inner rectangles of a staircase is also denoted n as in the case of a rectangular drawing. Fig.2 shows a staircase with $n = 6$ and $m = 5$. Note that a rectangular drawing is a staircase with $m = 1$ by definition.

2.2 Deletable Rectangles

The *deletable rectangle* of a staircase is the uppermost rectangle among the rectangles satisfying the following four conditions:

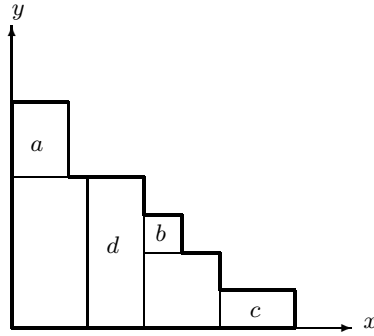


Fig. 2. A staircase with $n = 6$ and $m = 5$

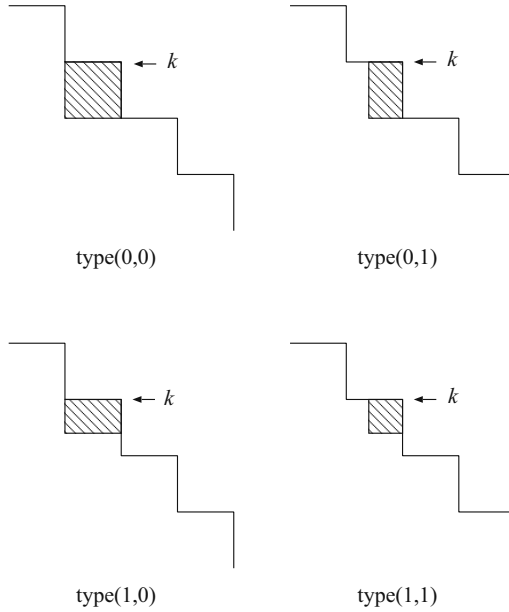


Fig. 3. The four types of deletable rectangles

1. Its top edge is contained in the border.
2. Its right edge is contained in the border.
3. The upward ray from the top left corner does not include any right side of the other rectangles.
4. The rightward ray from the bottom right corner does not include any top side of the other rectangles.

In the staircase in Fig.2, b is the deletable rectangle. Rectangles b and c satisfy the above four conditions. Note that rectangle a does not satisfy condition 4:

The rightward ray from the bottom right corner of a includes the top side of rectangle d . It is easy to see that the deletable rectangle is uniquely defined for every staircase.

Deletable rectangles are classified into the following four types as shown in Fig.3. Let r be a deletable rectangle whose top side is on the k -th step.

- type(0,0) : The k -th step of the staircase coincides with the top side of r and the right side of r intersects the $(k-1)$ -step. The deletion of r decreases the number of the steps of the staircase by one.
- type(0,1) : The top side of r is strictly included in the k -th step of the staircase and the right side of r intersects the $(k-1)$ -th step. The deletion of r does not change the number of the steps of the staircase.
- type(1,0) : The k -th step of the staircase coincides with the top side of r and the right side of r does not intersect the $(k-1)$ -step. The deletion of r does not change the number of the steps of the staircase.
- type(1,1) : The top side of r is strictly included in the k -th step of the staircase and the right side of r intersects the $(k-1)$ -th step. The deletion of r increases the number of the steps of the staircase by one.

3 A $(4n - 4)$ -Bit Representation of a Rectangular Drawing

3.1 A String Representation of a Staircase

Let S_n be a staircase with n rectangles. The configuration obtained by deleting the deletable rectangle r_n from S_n is also a staircase since r_n satisfies conditions 1 and 2. Then let the resultant staircase be S_{n-1} . Again deleting the deletable rectangle r_{n-1} from S_{n-1} , we obtain a new staircase S_{n-2} . In this way, we obtain staircases $S_{n-3}, \dots, S_1$, where S_1 is the staircase with $n = 1$, that is, a single rectangle.

Since the deletable rectangle r_i is uniquely defined for staircase S_i , the sequence $S_n, \dots, S_1$ is also unique to S_n . The idea of our representation is to construct staircase $S_1, \dots, S_n$ by adding rectangle r_i , which is the deletable rectangle of S_i , to staircase S_{i-1} .

Consider adding a rectangle r to a staircase with m steps and constructing a new one. If the top edge of r is level with one of the steps of the staircase, r will be the deletable rectangle of type(0,1) or type(1,1) in the resultant staircase. Otherwise, r will be of type(0,0) or type(1,0).

We define the *addition position ell* of the top edge of added rectangle r as follows:

- $\ell = 2i$ if the top edge of r is level with the i -th step of the staircase;
- $\ell = 2i - 1$ if the vertical position of the top edge of r is between those of the $(i-1)$ -th and the i -th steps;
- $\ell = 2m + 1$ if the top edge of r is above the m -th step of the staircase;

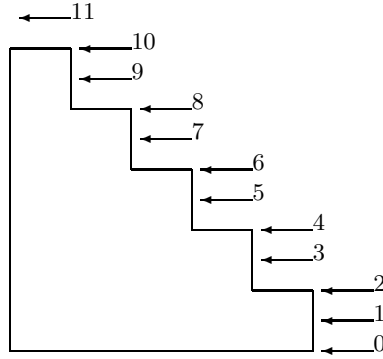


Fig. 4. The addition positions of a staircase with $m = 5$

– $\ell = 0$ if the top edge of r is on the x -axis. (This is defined for the sake of convenience).

Fig.4 shows the twelve reference positions $0, \dots, 11$ of a staircase with $m = 5$

Now we can describe sequence of stairs $S_1, \dots, S_n$ by using addition positions and types of the $n - 1$ deletable rectangles. Consider the example of a rectangular drawing in Fig.5, where rectangle i is the deletable rectangle of the staircase S_i . Then

1. S_2 is obtained by adding a rectangle of type(0,0) to S_1 at addition position 1;
2. S_3 is obtained by adding a rectangle of type(0,1) to S_2 at addition position 4;
3. S_4 is obtained by adding a rectangle of type(1,0) to S_3 at addition position 5;
4. S_5 is obtained by adding a rectangle of type(0,0) to S_4 at addition position 3;
5. S_6 is obtained by adding a rectangle of type(1,1) to S_5 at addition position 4;
6. S_7 is obtained by adding a rectangle of type(1,1) to S_6 at addition position 4.

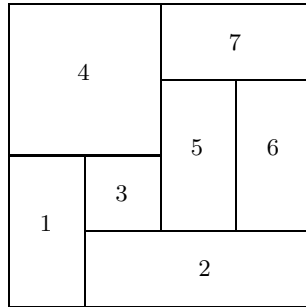


Fig. 5. A rectangular drawing represented by 1000001110001100011

First the above description is represented as string w on alphabet $\{0, A, B\}$ by the following procedure:

Procedure: ENCODE

1. Let p_i be the addition position appearing in line i of the above description. Conventionally, let $p_0 = 1$.
2. For $i = 1, \dots, n-1$, let ℓ_i be $p_i - p_{i-1}$.
3. According to the type of rectangle appearing in line i of the above description, modify ℓ_i as follows:
 - type(0,0): $\ell_i \leftarrow \ell_i + 1$;
 - type(0,1): $\ell_i \leftarrow \ell_i + 2$;
 - type(1,0): $\ell_i \leftarrow \ell_i + 2$;
 - type(1,1): $\ell_i \leftarrow \ell_i + 3$;
4. According to the type of rectangle appearing in line i of the above description, set symbol $a_i \in \{A, B\}$ as follows:
 - type(0,0): $a_i \leftarrow A$;
 - type(0,1): $a_i \leftarrow A$;
 - type(1,0): $a_i \leftarrow B$;
 - type(1,1): $a_i \leftarrow B$;
5. $w \leftarrow 0^{\ell_1} a_1 0^{\ell_2} a_2 \dots 0^{\ell_{n-1}} a_{n-1}$.

For the rectangular drawing in Fig.5, $w = A0000A000BA00B000B$ is obtained.

Note that the value of ℓ_i is nonnegative and w is well-defined. For example, assume that the rectangle r in line $i-1$ is of type(0,0) and its addition position is k . Then the addition position of the rectangle s appearing in line i must be at least $k-1$: Otherwise r would be the deletable rectangle in S_i . Therefore, the value ℓ_i computed in the step 2 of the procedure is at least -1 and it is modified to a nonnegative value in step 3. For the other types of rectangles, similar properties hold.

The string w holds complete piece of information for reconstructing staircase S_n . The following procedure **DECODE** reconstructs the staircase from w .

Procedure: DECODE

1. Let S_1 be a single rectangle. $\ell \leftarrow 1$.
2. Read symbols of w from head to tail. According to the symbol, do the following operations:
 - 0: $\ell \leftarrow \ell + 1$.
 - A : If ℓ is odd, add a rectangle of type(0,0) at addition position ℓ and $\ell \leftarrow \ell - 1$. If ℓ is even, add a rectangle of type(0,1) at addition position ℓ and $\ell \leftarrow \ell - 2$.
 - B : If ℓ is odd, add a rectangle of type(1,0) at addition position ℓ and $\ell \leftarrow \ell - 2$. If ℓ is even, add a rectangle of type(1,1) at addition position ℓ and $\ell \leftarrow \ell - 3$.

3.2 A $(4n - 4)$ -Bit Representation of a Rectangular Drawing

Consider a string representation $w \in \{0, A, B\}^*$ for a rectangular drawing. When w is decoded by **DECODE**, the final value of ℓ is determined on the type of the last-added rectangle r :

- $\ell = 0$ if r is of type(0,1) and
- $\ell = 1$ if r is of type(1,0) or type(1,1).

Then, if r is of type(0,1), append one 0 to the tail of w so that procedure **DECODE** always terminates with $k = 1$. Now the following theorem holds.

Theorem 1. *Let $w \in \{0, A, B\}^*$ be a string representing a rectangular drawing with n rectangles. Then*

- (1) *Symbols A and B collectively appear exactly $n - 1$ times in w .*
- (2) *Symbol 0 appears exactly $2n - 2$ times in w .*

Proof. (1) is trivial and we prove (2).

During the execution of **DECODE**, let m be the number of steps of the current staircase and ℓ be the current addition position. Then let $\phi = 2m + 1 - \ell$. At the end of step 1, $\phi = 2$ since $m = 1$ and $\ell = 1$. In step 2, ϕ changes according to the type of the added rectangle as follows:

Type of rectangle	Δm	Δk	$\Delta \phi$
type(0,0)	+1	-1	+3
type(0,1)	0	-2	+2
type(1,0)	0	-2	+2
type(1,1)	-1	-3	+1

Procedure **DECODE** starts with a single rectangle S_1 and ends with a rectangular drawing S_n . Both S_1 and S_n are staircases with one step. Therefore, the numbers of added rectangles of type(0,0) and type(1,1) must be the same since adding a rectangle of type(0,0) increases m by one and adding a rectangle of type(1,1) decreases m by one. Then $n - 1$ added rectangles contribute $2(n - 1)$ to the value of ϕ .

On the other hand, **DECODE** ends with $m = \ell = 1$, that is, $\phi = 2$. Since one 0 decreases ϕ by one, the number of 0's in w is $2 + 2(n - 1) - 2 = 2(n - 1)$. $\square$

By replacing A and B with 10 and 11, respectively, we obtain the following.

Corollary 1. *A rectangular drawing with n rectangles is represented in $4n - 4$ bits.*

4 An Alternative Proof That $R(n) \leq 13.5^{n-1}$

Amano et al. showed that $R(n)$ is asymptotically approximated by a geometric progression, that is, there exists $c = \lim_{n \rightarrow \infty} R(n)^{1/n}$ and $11.56 < c < 28.3$ [8]. Fujimaki et al. proved a recurrence for $R(n)$, which was used to show $R(n) \leq 13.5^{n-1}$ as follows [6].

Theorem 2. $R(n) = S'(n, 1, 1)$, where $S'(n, m, \ell)$ is given by the following recurrence:

If $\ell > 2m$,

$$S'(n, m, \ell) = S'(n, m, 2m). \quad (1)$$

If $\ell \leq 2m$ and ℓ is odd, then

$$\begin{aligned} & S'(n, m, 2k - 1) \\ &= S'(n, m, 2k - 2) \\ &+ S'(n - 1, m - 1, 2k) \\ &+ S'(n - 1, m, 2k + 1), \end{aligned} \quad (2)$$

where $\ell = 2k - 1$.

If $\ell \leq 2m$ and ℓ is even, then

$$\begin{aligned} & S'(n, m, 2k) \\ &= S'(n, m, 2k - 1) \\ &+ S'(n - 1, m, 2k + 2) \\ &+ S'(n - 1, m + 1, 2k + 3), \end{aligned} \quad (3)$$

where $\ell = 2k$.

The boundary condition is

$$S'(n, m, \ell) = \begin{cases} 0 & \text{if } m > n, m < 1, \text{ or } \ell < 1, \\ 1 & \text{if } n = m = \ell = 1. \end{cases} \quad (4)$$

By analyzing the recursion tree of $S'(n, 1, 1)$, the following inequality is obtained [6].

$$R(n) \leq 2^{n-1} \binom{3n-3}{n-1} = 2^{n-1} \cdot \frac{(3n-3)!}{(n-1)!(2n-2)!}. \quad (5)$$

By mathematical induction,

$$R(n) \leq 13.5^{n-1} \quad (6)$$

is obtained.

The above inequality (5) is a direct consequence of theorem 1, that is, a rectangular drawing with n rectangles is represented as string $w \in \{0, A, B\}^*$ such that $n - 1$ out of $3n - 3$ symbols of w are A or B . Now we have an alternative proof for equation (6).

5 Concluding Remarks

In this paper, a $(4n - 4)$ -bit representation of a rectangular drawing is introduced. This representation gives an alternative proof that $R(n) \leq 13.5^{n-1}$. For

asymptotical behavior of $R(n)$, see [6,8]. A polynomial time algorithm computing $R(n)$ and a recurrence were published in [7].

We would like to thank an unknown referee, who commented that string $w \in \{0, A, B\}$ can be encoded in $(n - 1) \times (\log_2 13.5) \simeq 3.75(n - 1)$ bits using standard deta compression techniques. Note that the $(4n - 4)$ -bit representation in the paper is a simple Huffman coding.

References

1. Yao, B., Chen, H., Cheng, C.K., Graham, R.: Floorplan Representations: Complexity and Connections. *ACM Trans. on Design Automation of Electronic Systems* 8(1), 55–80 (2003)
2. Zhuang, C., Sakanushi, K., Jin, L., Kajitani, Y.: An Extended Representation of Q-sequence for Optimizing Channel-Adjacency and Routing-Cost. In: *The 2003 Conference on Asia South Pacific Design Automation*, pp. 21–24 (2003)
3. Zhao, H.A., Liu, C., Kajitani, Y., Sakanushi, K.: EQ-Sequences for Coding Floorplans. *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences* E87-A(12), 3244–3250 (2004)
4. Fujimaki, R., Takahashi, T.: A Surjective Mapping from Permutations to Room-to-Room Floorplans. *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences* E90-A(4), 823–828 (2007)
5. Yamanaka, K., Nakano, S.: Coding Floorplans with Fewer Bits. *IEICE Trans. Fundamentals* E89-A(5), 1181–1185 (2006)
6. Fujimaki, R., Inoue, Y., Takahashi, T.: An Asymptotic Estimate of the Numbers of Rectangular Drawings or Floorplans. In: *The 2009 IEEE International Symposium on Circuits and Systems* (2009)
7. Inoue, Y., Fujimaki, R., Takahashi, T.: Counting Rectangular Drawings or Floorplans in Polynomial Time. *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences* E92-A(4), 1115–1120 (2009)
8. Amano, K., Nakano, S., Yamanaka, K.: On the Number of Rectangular Drawings: Exact Counting and Lower and Upper Bounds. *IPSJ SIG Notes*, 2007-AL-115-5, pp. 33–40 (2007)

Relationship between Approximability and Request Structures in the Minimum Certificate Dispersal Problem^{*}

Tomoko Izumi¹, Taisuke Izumi², Hirotaka Ono³, and Koichi Wada²

¹ College of Information Science and Engineering, Ritsumeikan University,
Kusatsu, 525-8577 Japan

izumi-t@fc.ritsumei.ac.jp

² Graduate School of Engineering, Nagoya Institute of Technology, Nagoya, 466-8555, Japan
{t-izumi,wada}@nitech.ac.jp

³ Graduate School of Information Science and Electrical Engineering, Kyushu University,
Fukuoka, 819-0395, Japan
ono@csce.kyushu-u.ac.jp

Abstract. Given a graph $G = (V, E)$ and a set $R \subseteq V \times V$ of requests, we consider to assign a set of edges to each node in G so that for every request (u, v) in R the union of the edge sets assigned to u and v contains a path from u to v . The *Minimum Certificate Dispersal Problem* (MCD) is defined as one to find an assignment that minimizes the sum of the cardinality of the edge set assigned to each node. In this paper, we give an advanced investigation about the difficulty of MCD by focusing on the relationship between its (in)approximability and request structures. We first show that MCD with general R has $\Theta(\log n)$ lower and upper bounds on approximation ratio under the assumption $P \neq NP$, where n is the number of nodes in G . We then assume R forms a clique structure, called *Subset-Full*, which is a natural setting in the context of the application. Interestingly, under this natural setting, MCD becomes to be 2-approximable, though it has still no polynomial time approximation algorithm whose factor better than $677/676$ unless $P = NP$. Finally, we show that this approximation ratio can be improved to $3/2$ for undirected variant of MCD with *Subset-Full*.

1 Introduction

Background and Motivation. Let $G = (V, E)$ be a directed graph and $R \subseteq V \times V$ be a set of ordered pairs of nodes, which represents requests about reachability between two nodes. For given G and R , we consider an assignment of a set of edges to each node in G . The assignment satisfies a request (u, v) if the union of the edge sets assigned to u and v contains a path from u to v . The *Minimum Certificate Dispersal Problem* (MCD) is the one to find the assignment satisfying all requests in R that minimizes the sum of the cardinality of the edge set assigned to each node.

^{*} This work is supported in part by KAKENHI no. 19700058, 21500013 and 21680001, Asahi-glass Foundation, Inamori Foundation and Hori Information Science Promotion Foundation.

This problem is motivated by a requirement in public-key based security systems, which are known as a major technique for supporting secure communication in a distributed system [3,5,6,7,8,10,11]. The main problem of the systems is to make each user's public key available to others in such a way that its authenticity is verifiable. One of well-known approaches to solve this problem is based on public-key certificates. A public-key certificate contains the public key of a user v encrypted by using the private key of a user u . Any user who knows the public key of u can use it to decrypt the certificate from u to v for obtaining the public key of v . All certificates issued by users in a network can be represented by a certificate graph: Each node corresponds to a user and each directed edge corresponds to a certificate. When a user w has communication request to send messages to a user v securely, w needs to know the public key of v to encrypt the messages with it. To compute v 's public-key, w uses a set of certificates stored in w and v in advance. Therefore, in a certificate graph, if a set of certificates stored in w and v contains a path from w to v , then the communication request from w to v is satisfied. In terms of cost to maintain certificates, the total number of certificates stored in all nodes must be minimized for satisfying all communication requests.

While, from the practical aspect, MCD should be handled in the context of distributed computing theory, its inherent difficulty as an optimization problem is not so clear even in centralized settings: Jung et al. discussed MCD with a restriction of available paths in [8] and proved that the problem is NP-hard. In their work, to assign edges to each node, only the restricted paths which are given for each request is allowed to be used. In [11], MCD, with no restriction of available paths, is proved to be also NP-hard even if the input graph is strongly connected. Known results about the complexity of MCD are actually only these NP-hardness. This fact yields a theoretical interest of revealing the (in)approximability of MCD. As for the positive side, MCD is polynomially solvable for bidirectional trees, rings and Cartesian products of graphs [11].

This paper also investigates how the request structures affect the difficulty of MCD. As seen above, MCD is doubly structured in a sense: One structure is the graph G itself and the other is the request structure R . We would like to investigate how the tractability of MCD changes as the topology of R changes. On MCD, our interest here is to investigate whether the hardness (of approximation) of MCD depends on the restrictions about R . This is a natural question not only from the theoretical viewpoint but also from the practical viewpoint, because, in public-key based security systems, a set of requests should have a certain type of structures. For example, it is reasonable to consider the situation in which a set of nodes belonging to a certain community should have requests between each other in the community. This situation is interpreted that R forms a clique structure. Thus the following question arises: If R forms a clique, can the approximability of MCD be improved?

Our Contribution. In this paper, we investigate the approximability of MCD from the perspective how the structure of R affects the complexity of MCD. We classify the set R of requests according to the elements of R : R is *subset-full* if for a subset V' of V , R consists of all reachable pairs of nodes in V' , and R is *full* if the subset V' is equal to V . Note that Subset-Full corresponds to the situation that R forms a clique. Table 1 summarizes the results in this paper.

Table 1. Approximability / Inapproximability shown in this paper

	Restriction on request		
	Arbitrary	Subset-Full	Full
Inapproximability	$\Omega(\log n)$	677/676	open
	261/260 (for bidirectional graphs)		
Approximation ratio	$O(\log n)$	2	2 [11]
		3/2 (for undirected graphs)	

Here we review our contribution. We first consider the general case: We show that if we have no restriction about R , a lower bound on approximation ratio for MCD is $\Omega(\log n)$ and an upper bound is $O(\log n)$, where n is the number of nodes. Namely, the lower and upper bounds coincide as $\Theta(\log n)$ in terms of order. Moreover, it is proved that we can still obtain the inapproximability $\Omega(1)$ of MCD even when the graph class is restricted to bidirectional graphs. As the second half of the contribution, for subset-full requests, we show that the lower bound of approximation ratio for MCD is 677/676 and the upper bound is 2. The upper bound is proved by a detailed analysis of the algorithm MinPivot, which is proposed in [11]. While Zheng et al. have shown that MinPivot achieves approximation ratio 2 with full requests, we can obtain the same approximation ratio by a different approach even when the set of requests is subset-full. In addition, by extending the approach, it is also shown that MinPivot guarantees 3/2 approximation ratio for MCD of the undirected variant with subset-full requests.

The remainder of the paper is organized as follows. In Section 2, we define the Minimum Certificate Dispersal Problem (MCD). Section 3 presents inapproximability of MCD with general R and one with Subset-Full. The upper bound of MCD with general R and one with Subset-Full are shown in Sections 4 and 5 respectively. Section 6 concludes the paper. All the proofs are omitted due to space limitation.

2 Minimum Certificate Dispersal Problem

Let $G = (V, E)$ be a directed graph, where V and E are the sets of nodes and edges in G respectively. An edge in E connects two distinct nodes in V . The edge from node u to v is denoted by (u, v) . The numbers of nodes and edges in G are denoted by n and m , respectively (i.e., $n = |V|, m = |E|$). A sequence of edges $p(v_0, v_k) = (v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$ is called a *path* from v_0 to v_k of length k . A path $p(v_0, v_k)$ can be represented by a sequence of nodes $p(v_0, v_k) = (v_0, v_1, \dots, v_k)$. For a path $p(v_0, v_k)$, v_0 and v_k are called the source and destination of the path respectively. The length of a path $p(v_0, v_k)$ is denoted by $|p(v_0, v_k)|$. For simplicity, we treat a path as the set of edges on the path when no confusion occurs. A shortest path from u to v is the one whose length is the minimum of all paths from u to v , and the distance from u to v is the length of a shortest path from u to v , denoted by $d(u, v)$.

A *dispersal* D of a directed graph $G = (V, E)$ is a family of sets of edges indexed by V , that is, $D = \{D_v \subseteq E | v \in V\}$. We call D_v a local dispersal of v . A local dispersal D_v indicates the set of edges assigned to v . The *cost* of a dispersal D , denoted by $c.D$, is the sum of the cardinalities of all local dispersals in D (i.e., $c.D = \sum_{v \in V} |D_v|$). A request

is a reachable ordered pair of nodes in G . For a request (u, v) , u and v are called the source and destination of the request respectively. A set R of requests is *subset-full* if there exists a subset of V such that R consists of all reachable pairs of nodes in V' (i.e., $R = \{(u, v) | u \text{ is reachable to } v \text{ in } G, u, v \in V' \subseteq V\}$), and R is *full* if the subset V' is equal to V . We say a dispersal D of G *satisfies* a set R of requests if a path from u to v is included in $D_u \cup D_v$ for any request $(u, v) \in R$.

The *Minimum Certificate Dispersal Problem (MCD)* is defined as follows:

Definition 1 (Minimum Certificate Dispersal Problem (MCD))

INPUT: A directed graph $G = (V, E)$ and a set R of requests.

OUTPUT: A dispersal D of G satisfying R with minimum cost.

The minimum among costs of dispersals of G that satisfy R is denoted by $c_{\min}(G, R)$. For short, the cost $c_{\min}(G, R)$ is also denoted by $c_{\min}(G)$ when R is full. Let D^{Opt} be an optimal dispersal of G which satisfies R (i.e., D^{Opt} is one such that $c(D^{Opt}) = c_{\min}(G, R)$).

In this paper, we deal with MCD for undirected graphs in Section 5.3. For an undirected graph G , the edge between nodes u and v is denoted by (u, v) or (v, u) . When an edge (u, v) is included in a local dispersal D_v , the node v has two paths from u to v and from v to u .

3 Inapproximability

It was shown in [11] that MCD for strongly connected graphs is NP-hard by a reduction from the VERTEX-COVER problem. In this section, we provide another proof of NP-hardness of MCD for strongly connected graphs, which implies a stronger inapproximability. Here, we show a reduction from the SET-COVER problem. For a collection C of subsets of a finite universal set U , $C' \subseteq C$ is called a *set cover* of U if every element in U belongs to at least one member of C' . Given C and a positive integer k , SET COVER is the problem of deciding whether a set cover $C' \subseteq C$ of U with $|C'| \leq k$ exists.

The reduction from SET-COVER to MCD is as follows: Given a universal set $U = \{1, 2, \dots, n\}$ and its subsets $S_1, S_2, \dots, S_m$ and a positive integer k as an instance I of SET-COVER, we construct a graph G_I including gadgets that mimic (a) elements, (b) subsets, and (c) a special gadget: (a) For each element i of the universe set $U = \{1, 2, \dots, n\}$, we prepare an element gadget u_i (it is just a vertex); let V_U be the set of element vertices, i.e., $V_U = \{u_i \mid i \in U\}$. (b) For each subset $S_j \in C$, we prepare a directed path $(v_{j,1}, v_{j,2}, \dots, v_{j,p})$ of length $p - 1$, where p is a positive integer used as a parameter. The end vertex $v_{j,p}$ is connected to the element gadgets that correspond to elements belonging to S_j . For example, if $S_1 = \{2, 4, 5\}$, we have directed edges $(v_{1,p}, u_2)$, $(v_{1,p}, u_4)$ and $(v_{1,p}, u_5)$. (c) The special gadget just consists of a base vertex r . This r has directed edges to all $v_{j,1}$'s of $j = 1, 2, \dots, m$. Also r has an incoming edge from each u_i . See Figure 1 as an example of the reduction, where $S_1 = \{1, 2, 3\}$, $S_2 = \{2, 4, 5\}$ and $S_3 = \{3, 5, 6\}$. We can see that G_I is strongly connected. The set R of requests contains the requests from the base vertex r to all element vertices u_i , i.e., $R = \{(r, u_i) \mid u_i \in V_U\}$.

We can show the following, although we omit the proof because it is straightforward:

(i) If the answer of instance I of SET-COVER is yes, then $c_{\min}(G, R) \leq pk + n$. (ii)

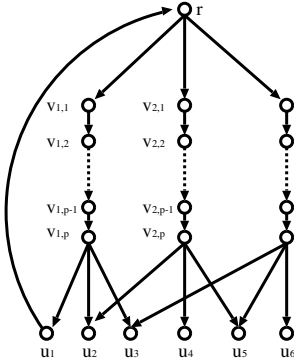


Fig. 1. Reduction for general case (from SET-COVER)

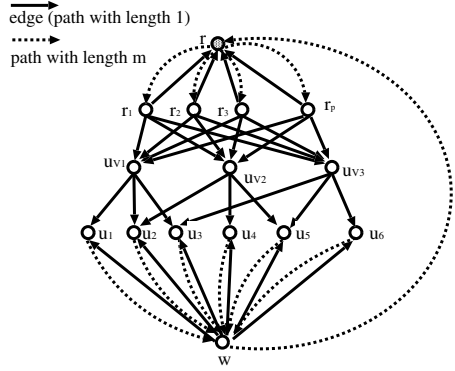


Fig. 2. Reduction for Subset-Full (from VERTEX-COVER)

Otherwise, $c_{\min}(G, R) \geq p(k+1) + n$. About the inapproximability of SET-COVER, it is known that SET-COVER has no polynomial-time approximation algorithm with factor better than $0.2267 \ln n$, unless $P = NP$ [1]. From this inapproximability, we obtain a *gap-preserving reduction* [2] as follows:

Lemma 1. *The above construction of G_I is a gap-preserving reduction from SET-COVER to MCD for strongly connected graphs such that*

- (i) *if $OPT_{SC}(I) \leq g(I)$, then $c_{\min}(G, R) \leq p \cdot g(I) + n$,*
- (ii) *if $OPT_{SC}(I) \geq g(I) \cdot c \ln n$, then $c_{\min}(G, R) \geq (p \cdot g(I) + n) \left(c \ln n - \frac{cn \ln n - n}{p \cdot g(I) + n} \right)$,*

where $OPT_{SC}(I)$ denotes the optimal value of SET-COVER for I and $c = 0.2267$.

By taking p large so as to satisfy $p \cdot g(I) + n = n^{1+\alpha}$ for $\alpha > 0$, we have the following:

Theorem 1. *There exists no $(0.2267(1 + \alpha)^{-1} \ln |V| - \varepsilon)$ factor approximation polynomial time algorithm of MCD for strongly connected graphs unless $P = NP$, where α and ε are arbitrarily small positive constants.*

We can obtain some inapproximability result for bidirectional graphs, by slightly modifying the graph G_I , though we omit the details.

Theorem 2. *There exists no $(261/260 - \varepsilon)$ factor approximation polynomial time algorithm of MCD for bidirectional graphs unless $P = NP$, where ε is an arbitrarily small positive constant.*

Again we consider another reduction from VERTEX-COVER for graphs with degree at most 4, in which we embed an instance to MCD problem with a subset-full request structure. As well as the reduction from SET-COVER, we prepare (a) edge gadgets, (b) vertex gadgets, and (c) special gadgets. The reduction from VERTEX-COVER to MCD with subset-full requests is as follows: Given $G = (V, E)$ with degree at most 4 and a positive integer k as an instance I of VERTEX-COVER, where $V = \{1, 2, \dots, n\}$ is the

vertex set and $E = \{e_1, e_2, \dots, e_m\}$ is the edge set, we construct an MCD graph G'_I . (a) For each edge e_i in E , we prepare an m -length directed path $(u_i, u_{i,1}, \dots, u_{i,m-1}, w)$ and (w, u_i) as an edge gadget, where w is a common vertex among edge gadgets. (b) For each vertex $j \in V$, we prepare a vertex u_j^V as a vertex gadget. If j is connected with edge e_i , we add directed edges (u_j^V, u_i) . For example, if $e_4 = \{2, 3\}$, we have directed edges (u_2^V, u_4) , (u_3^V, u_4) . Note that each u_i has exactly two incoming edges from vertex gadgets. (c) The special gadgets consist of p base vertices $r_1, r_2, \dots, r_p$ and one root vertex r . Each r_j and r are connected by path $(r, r_{j,1}, \dots, r_{j,m-1}, r_j)$ and edge (r_j, r) . Also, each r_i has directed edges to all u_j^V 's of $j = 1, 2, \dots, m$. Furthermore, we prepare an m -length directed path from w to r , i.e., $(w, w_1, \dots, w_{m-1}, r)$. See Figure 2 as an example of the reduction, in which we have $e_2 = \{1, 2\}$, $e_3 = \{1, 3\}$ and $e_5 = \{2, 3\}$. We can see that G'_I is strongly connected. The set R' of requests are defined as $R' = R_{a,a} \cup R_{a,c} \cup R_{c,c}$, where $R_{a,a} = \{(u_i, u_j) \mid i, j = 1, 2, \dots, m, \text{ and } i \neq j\}$, $R_{a,c} = \{(u_i, r_j), (r_j, u_i) \mid i = 1, \dots, m\}$ and $R_{c,c} = \{(r_i, r_j) \mid i, j = 1, 2, \dots, p, \text{ and } i \neq j\}$.

Lemma 2. *Let $p = m$. The above construction of G'_I and R' is a gap-preserving construction from VERTEX-COVER with degree at most 4 to MCD with subset-full requests for strongly connected graphs such that:*

- (i) *If $\text{OPT}_{VC}(I) = g(I)$, then $c_{\min}(G'_I, R') \leq m(g(I) + 3m + 3)$.*
- (ii) *If $\text{OPT}_{VC}(I) > c \cdot g(I)$, then $c_{\min}(G'_I, R') > m(g(I) + 3m + 3)(c - \frac{(3m+3)(c-1)}{g(I)+3m+3})$,*

where $\text{OPT}_{VC}(I)$ denotes the optimal value of VERTEX-COVER for I and $c = 53/52$.

The constant $c = 53/52$ represents an inapproximability bound for VERTEX-COVER with degree at most 4 under the assumption $P \neq NP$ [4]. From this lemma and $4g(I) \geq m$, we obtain the following theorem:

Theorem 3. *There exists no $(677/676 - \varepsilon)$ factor approximation polynomial time algorithm of MCD with subset-full requests for strongly connected graphs unless $P = NP$, where ε is an arbitrarily small positive constant.*

4 Approximability

In the previous section, we show that it is difficult to design a polynomial time approximation algorithm of MCD whose factor is better than $(0.2267(1 + \alpha)^{-1} \ln n - \varepsilon)$, even if we restrict that the input graph is strongly connected. In this section, in contrast, we show that MCD has a polynomial time approximation algorithm whose factor is $O(\log n)$, which is applicable for general graphs. This implies that we clarify an optimal approximability / inapproximability bound in terms of order under the assumption $P \neq NP$.

The idea of $O(\log n)$ -approximation algorithm is based on formulating MCD as a *submodular set cover problem* [9]: Let us consider a finite set N , a nonnegative cost function c_j associated with each element $j \in N$, and non-decreasing submodular function $f : 2^N \mapsto \mathbb{Z}^+$. A function f is called *non-decreasing* if $f(S) \leq f(T)$ for $S \subseteq T \subseteq N$, and is called *submodular* if $f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$ for $S, T \subseteq N$. For a subset $S \subseteq N$, the cost of S , say $c(S)$, is $\sum_{j \in S} c_j$.

By these f , c and N , the submodular set cover problem is formulated as follows:

[Minimum Submodular Set Cover (SSC)]

$$\min \left\{ \sum_{j \in S} c_j : f(S) = f(N) \right\}.$$

It is known that the greedy algorithm of SSC has approximation ratio $H(\max_{j \in N} f(j))$ where $H(i)$ is the i -th harmonic number if f is integer-valued and $f(\emptyset) = 0$ [9]. Note that $H(i) < \ln i + 1$.

We here claim that our problem is considered a submodular set cover problem. Let $N = \bigcup_{u \in V} \{x_{e,u} \mid e \in E\}$. Intuitively, $x_{e,u} \in S \subseteq N$ represents that the local dispersal of u contains $e \in E$ in S , i.e., $e \in D_u$. For $S \subseteq N$, we define $d_S(u, v)$ as the distance from u to v under the setting that each edge $e \in D_u \cup D_v$ of S has length 0 otherwise 1. That is, if all edges are included in $D_u \cup D_v$ of S , then $d_S(u, v) = 0$. If no edge is included in $D_u \cup D_v$ of S , then $d_S(u, v)$ is the length of a shortest path from u to v of G . Let $f(S) = \sum_{(u,v) \in R} (d_\emptyset(u, v) - d_S(u, v))$. This f is integer-valued and $f(\emptyset) = 0$. In the problem setting of MCD, we can assume that for any $(u, v) \in R$, G has a (directed) path from u to v . (Otherwise, we have no solution). Then the condition $f(N) = f(S)$ means that all the requests are satisfied. Also cost c reflects the cost of MCD.

Then we have the following lemma:

Lemma 3. *Function f defined as above is a non-decreasing submodular function.*

Notice that f can be computed in polynomial time.

By these, MCD is formulated as a submodular set cover problem. Since we have $\max_{x_{e,u} \in N} f(\{x_{e,u}\}) \leq |R| \max_{u,v} d_\emptyset(u, v) \leq n^3$, the approximation ratio of the greedy algorithm is $O(\log n)$. We obtain the following.

Theorem 4. *There is a polynomial time algorithm with approximation factor $O(\log n)$ for MCD.*

5 Approximation Algorithm for Subset-Full

Zheng et al. have proposed a polynomial-time algorithm for MCD, called MinPivot, which achieves approximation ratio 2 for strongly connected graphs when a set R of requests is full. In this section, we show that even when R is subset-full, MinPivot achieves approximation ratio 2 for strongly connected graphs. Moreover, we show that MinPivot is a $3/2$ -approximation algorithm for MCD of the undirected variant with subset-full requests.

5.1 Algorithm MinPivot

A pseudo-code of the algorithm MinPivot is shown in Algorithm 1.¹ For the explanation of the algorithm, we define $\mathcal{P}(u, v)$ as the minimum-cardinality set of edges that constitute a round-trip path between u and v on G .

¹ Although the original MinPivot is designed to work for any set of requests, we here show a simplified one because we focus on the case when R is subset-full.

Algorithm 1. MinPivot ($G = (V, E), R$)

```

1:  $V' := \{v, w \in V \mid (v, w) \in R\}$ 
2: for all  $u \in V$  do
3:   for all  $v \in V'$  do
4:      $D_v := \mathcal{P}(u, v)$  and  $D(u) := \{D_v \mid v \in V\}$ 
5:   end for
6: end for
7: output  $\min_{u \in V} \{c \cdot D(u)\}$  and its  $D(u)$ .
```

In dispersals returned by MinPivot, one node is selected as a *pivot*. Each request is satisfied by a path via the selected pivot. The algorithm works as follows: It picks up a node u as a candidate of the pivot. Then, for nodes v, w in each request $(v, w) \in R$, MinPivot stores a round-trip path between v (reps. w) and the pivot u in D_v (resp. D_w) such that the sum of edges included in the round-trip path is minimum. Since there is a path from v to w via the pivot u in $D_v \cup D_w$ for each request (v, w) , the dispersal satisfies R . For every pivot candidate, the algorithm MinPivot computes the corresponding dispersal and returns the minimum-cost one among all computed dispersals.

In [11], the following theorem is proved.

Theorem 5. *For a strongly connected graph G , MinPivot is a 2-approximation algorithm for MCD on G with a full request. It completes in $O(n^7)$ time for a strongly connected graph and in $O(nm)$ time for an undirected graph.*

5.2 Proof of 2-Approximation for Strongly Connected Graphs

In this subsection, we prove the following theorem.

Theorem 6. *For a strongly connected graph G and a subset-full request R , MinPivot is a 2-approximation algorithm.*

We first introduce several notations used in the proof: The set of nodes included in requests in R is denoted by V_R , that is, $V_R = \{u, v \mid (u, v) \in R\}$. Let x be a node in V_R with the minimum local dispersal in D^{Opt} (i.e., $|D_x^{Opt}| = \min\{|D_v^{Opt}| \mid v \in V_R\}$). When there is more than one node with the minimum local dispersal, x is defined as one of them chosen arbitrarily. In the following argument, we can consider only the case of $|D_x^{Opt}| > 0$: If $|D_x^{Opt}|$ is zero, any node in V_R must have two paths from/to x in its local dispersal to satisfy the requests for x . Then, the optimal solution is equivalent to that computed by MinPivot whose pivot candidate is x , which implies that MinPivot returns an optimal solution. Let D^{MP} denote an output of the algorithm MinPivot. The following proposition clearly holds.

Proposition 1. *For a dispersal D , if there exists a node u such that the local dispersal D_v of any node v in V_R contains a round-trip path between v and u , then $c \cdot D^{MP} \leq c \cdot D$.*

The idea of the proof is that we construct a feasible dispersal D with cost at most $2 \cdot c \cdot D^{Opt}$, which satisfies the condition shown in Proposition 1. It follows that the cost

of the solution by MinPivot is bounded by $2 \cdot c \cdot D^{Opt}$. We construct the dispersal D from D^{Opt} by additionally giving the minimum-size local dispersal to all nodes in V_R . More precisely, for every node $v \in V_R$, $D_v = D_v^{Opt} \cup D_x^{Opt}$.

Theorem 6 is easily proved from the following lemma and Proposition 1.

Lemma 4. *In the dispersal D constructed in the above way, every node v in V_R has a round-trip path between v and x in D_v . In addition, $c \cdot D \leq 2 \cdot c \cdot D^{Opt}$ is satisfied.*

5.3 Proof of 3/2-Approximation for Undirected Graphs

In this subsection, we prove that the approximation ratio of MinPivot is improved for MCD of the undirected variant. That is, we prove the following theorem.

Theorem 7. *For an undirected graph G and a subset-full request R , MinPivot is a 3/2-approximation algorithm.*

In the proof, we take the same approach as the one of Theorem 6: We construct a dispersal D with cost at most $\frac{3}{2} \cdot c \cdot D^{Opt}$, which satisfies the condition in Proposition 1. Since Proposition 1 also clearly holds in undirected graphs, it follows that the cost of the solution by MinPivot is bounded by $\frac{3}{2} \cdot c \cdot D^{Opt}$. In the proof of Theorem 6, we show that when all the edges in D_x^{Opt} are added to the local dispersal of every node in V_R , the cost of the dispersal D is at most twice as much as that of the optimal dispersal. Our proof of Theorem 7 is based on the idea that we construct a dispersal D by adding each edge in D_x^{Opt} to at most $|V_R|/2$ local dispersals.

In what follows, we show the construction of D . We define a rooted tree T from an optimal dispersal D^{Opt} . To define T , we first assign a *weight* to each edge: To any edge in D_x^{Opt} , the weight zero is assigned. All the other edges are assigned the weight one. A rooted tree $T = (V, E_T)$ ($E_T \subseteq E$) is defined as a shortest path tree with root x (in terms of weighted graphs) that spans all the nodes in V_R . Let $p_T(u, v)$ be the shortest path from a node u to v on the tree T . The weight of a path $p(u, v)$ is defined by the total weight of the edges on the path and denoted by $w.p(u, v)$. For each node v , let $p_T(v, v) = \phi$ and $w.p_T(v, v) = 0$. From the construction of the tree $T = (V, E_T)$, we obtain that $\sum_{v \in V_R} w.p_T(x, v) < c \cdot D^{Opt}$.

For each edge e in D_x^{Opt} , let $C(e)$ be the number of nodes from which path to the node x on T includes the edge e : $C(e) = |\{v \in V_R \mid e \in p_T(x, v)\}|$. The construction of the desired dispersal depends on whether any edge e in D_x^{Opt} satisfies $C(e) \leq |V_R|/2$ or not.

In the case that $C(e) \leq |V_R|/2$ holds for any edge e in D_x^{Opt} , the dispersal D' is constructed in the following way: $D' = \{D'_v \mid v \in V\}$, where $D'_v = p_T(x, v)$ for node v in V_R , and $D'_v = \phi$ for node v in $V \setminus V_R$.

Lemma 5. $c \cdot D' \leq \frac{3}{2} \cdot c \cdot D^{Opt}$

We consider the case that there is an edge such that $C(e) > |V_R|/2$. Let T_v be the subtree of T induced by node v and all of v 's descendants, and $V(T_v)$ be a set of nodes in T_v . The set of edges in D_x^{Opt} such that $C(e) > |V_R|/2$ is denoted by $\hat{D}_x^{Opt}$. Let y be the node farthest from x of those adjacent to some edge in $\hat{D}_x^{Opt}$. A dispersal D'' is constructed such that every node in V_R has a path from itself to node y : $D'' = \{D''_v \mid v \in V\}$, where

$D'_v = p_T(y, v)$ for node v in $V_R \cap V(T_y)$, $D'_v = p_T(x, v) \cup p_T(x, y)$ for node v in $V_R \setminus V(T_y)$, and $D'_v = \emptyset$ for node v in $V \setminus V_R$.

Lemma 6. $c \cdot D'' \leq \frac{3}{2} \cdot c \cdot D^{Opt}$

From Lemmas 5 and 6, Theorem 7 is proved.

6 Concluding Remarks

In this paper, we investigate the (in)approximability of MCD from a perspective of how topological structures of R affect the complexity of MCD. While the approximability bound of MCD for a general setting of R is evaluated as $\Theta(\log n)$ under the assumption $P \neq NP$, MCD for Subset-Full is 2-approximable though it is still inapproximable within a small constant factor unless $P = NP$. The complexity of MCD for Full, which is a special case of Subset-Full, is still open. We actually conjecture that MinPivot returns an optimal solution for MCD with Full; if it is correct, we will obtain an interesting contrast similar to the relation between Minimum Steiner Tree and Minimum Spanning Tree.

References

1. Alon, N., Moshkovitz, D., Safra, S.: Algorithmic construction of sets for k -restrictions. *ACM Transactions on Algorithms* 2(2), 153–177 (2006)
2. Arora, S., Lund, C.: Hardness of approximation. In: Hochbaum, D. (ed.) *Approximation Algorithms for NP-hard problems*, pp. 399–446. PWS publishing company (1995)
3. Capkun, S., Buttyan, L., Hubaux, J.-P.: Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing* 2(1), 52–64 (2003)
4. Chlebík, M., Chlebíková, J.: Complexity of approximating bounded variants of optimization problems. *Theoretical Computer Science* 354(3), 320–338 (2006)
5. Gouda, M.G., Jung, E.: Certificate dispersal in ad-hoc networks. In: *Proceeding of the 24th International Conference on Distributed Computing Systems (ICDCS 2004)*, March 2004, pp. 616–623 (2004)
6. Gouda, M.G., Jung, E.: Stabilizing certificate dispersal. In: Tixeuil, S., Herman, T. (eds.) *SSS 2005*. LNCS, vol. 3764, pp. 140–152. Springer, Heidelberg (2005)
7. Hubaux, J., Buttyan, L., Capkun, S.: The quest for security in mobile ad hoc networks. In: *Proceeding of the 2nd ACM international symposium on Mobile ad hoc networking and computing (Mobihoc 2001)*, October 2001, pp. 146–155 (2001)
8. Jung, E., Elmallah, E.S., Gouda, M.G.: Optimal dispersal of certificate chains. In: Guerraoui, R. (ed.) *DISC 2004*. LNCS, vol. 3274, pp. 435–449. Springer, Heidelberg (2004)
9. Wolsey, L.A.: An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica* 2(4), 385–393 (1982)
10. Zheng, H., Omura, S., Uchida, J., Wada, K.: An optimal certificate dispersal algorithm for mobile ad hoc networks. *IEICE Transactions on Fundamentals* E88-A(5), 1258–1266 (2005)
11. Zheng, H., Omura, S., Wada, K.: An approximation algorithm for minimum certificate dispersal problems. *IEICE Transactions on Fundamentals* E89-A(2), 551–558 (2006)

Coordinate Assignment for Cyclic Level Graphs

Christian Bachmaier¹, Franz J. Brandenburg¹,
Wolfgang Brunner¹, and Raymund Fülöp²

¹ University of Passau, Germany

{bachmaier, brandenb, brunner}@fim.uni-passau.de

² Technische Universität München, Germany

fueloep@in.tum.de

Abstract. The Sugiyama framework is the most commonly used concept for visualizing directed graphs. It draws them in a hierarchical way and operates in four phases: cycle removal, leveling, crossing reduction, and coordinate assignment. However, there are situations where cycles must be displayed as such, e. g., distinguished cycles in the biosciences and scheduling processes which repeat in a daily or weekly turn. This excludes the removal of cycles. In their seminal paper Sugiyama et al. introduced recurrent hierarchies as a concept to draw graphs with cycles. However, this concept has not received much attention in the following years. In this paper we supplement our cyclic Sugiyama framework and investigate the coordinate assignment phase. We provide an algorithm which runs in linear time and constructs drawings which have at most two bends per edge and use quadratic area.

1 Introduction

The Sugiyama framework [9] is among the most intensively studied algorithms in graph drawing. It is the standard technique to draw directed graphs, and displays them in a hierarchical manner. It consists of the four phases of cycle removal, leveling, crossing reduction, and coordinate assignment. Typical applications are schedules, UML diagrams, and flow charts.

In its first phase the Sugiyama framework destroys all cycles. However, there are many situations where this is unacceptable. There are well-known cycles in the biosciences [7], where it is a common standard to display these cycles as such. Another inevitable use are repeating processes, such as daily, weekly, or monthly schedules which define the Periodic Event Scheduling Problem [8].

In their seminal paper [9], Sugiyama et al. proposed a solution for both the hierarchic and the cyclic style. The latter is called a *recurrent hierarchy* which is a level graph with additional edges from the last to the first level. It can be drawn in 2D where the levels are rays from a common center (see Fig. 1(a)) and each edge $e = (u, v)$ is a monotone counterclockwise poly-spiral segment from u to v wrapping around the center at most once. An alternative is a 3D drawing on a cylinder (see Fig. 1(c)). A combination would be the best of both worlds: an interactive 2D view with horizontal levels. It can be scrolled upwards

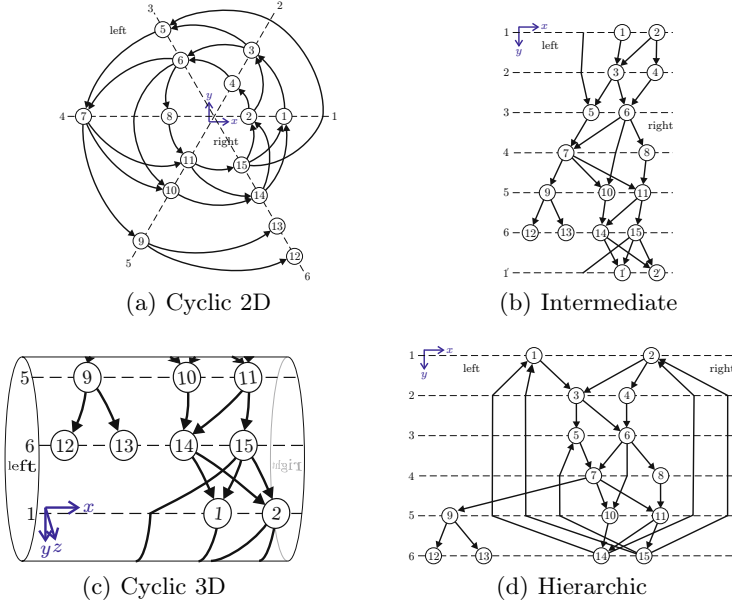


Fig. 1. Example drawings

and downwards infinitely and always shows a different part of the cylinder, see Fig. 1(b) for a snap shot, which also represents our intermediate drawing.

In cyclic drawings edges are irreversible and cycles are represented in a direct way. Thus, the cycle removal phase is not needed. This saves much effort, since the underlying feedback arc set problem is $\mathcal{NP}$ -hard [4]. Further advantages over hierarchic drawings (see Fig. 1(d)) are shorter edges and fewer crossings.

A planar recurrent hierarchy is shown on the cover of the textbook by Kaufmann and Wagner [6]. There it is stated that recurrent hierarchies are “unfortunately [...] still not well studied”. After investigating the leveling phase [1], we consider the coordinate assignment phase for the cyclic case. There are several algorithms for non-cyclic coordinate assignment [6]. We modify the established algorithm of Brandes and Köpf [3] for cyclic level graphs and provide a linear time algorithm using quadratic area and with at most two bends per edge.

2 Preliminaries

A *cyclic k -level graph* $G = (V, E, \phi)$ ($k \geq 2$) is a directed graph without self-loops with a given surjective level assignment of the vertices $\phi: V \rightarrow \{1, 2, \dots, k\}$. Let $V_i \subset V$ be the set of vertices v with $\phi(v) = i$. For two vertices $u, v \in V$ let $\text{span}(u, v) := \phi(v) - \phi(u)$ if $\phi(u) < \phi(v)$ and $\text{span}(u, v) := \phi(v) - \phi(u) + k$ otherwise. For an edge $e = (a, b) \in E$ we define $\text{span}(e) := \text{span}(a, b)$. An edge e with $\text{span}(e) = 1$ is *short*, otherwise *long*. A graph is *proper* if all edges are short. Each cyclic level graph can be made proper by adding $\text{span}(e) - 1$ dummy

vertices for each edge e and thus splitting e in $\text{span}(e)$ many short edges, which we call the *segments* of e . In total, this leads up to $\mathcal{O}(|E| \cdot k)$ new vertices. The first and the last segment of each edge are its *outer segments*, and all other segments between two dummy vertices are its *inner segments*. A proper cyclic k -level graph $G = (V, E, \phi, <)$ is *ordered* if $<$ is a total ordering for each V_i ($1 \leq i \leq k$). In accordance to [3] we say that in an ordered cyclic level graph there are two *conflicting* segments if they cross or share a vertex. Conflicts are of *type 0*, *1* or *2*, if they are induced by 0, 1, or 2 inner segments, respectively.

We represent drawings of cyclic level graphs in an *intermediate drawing* in the remainder of the paper assigning each vertex v two coordinates $x(v) \in \mathbb{R}$ and $y(v) = \phi(v) \in \mathbb{N}$. The x -coordinate increases from *left* to *right*, the y -coordinate increases *downwards* in edge direction, see Fig. 1(b). All vertices on level 1 are duplicated on level $k + 1$ using the same x -coordinates. Each segment $s = (u, v)$ is drawn straight-line from $(x(u), y(u))$ to $(x(v), y(u) + 1)$ with *slope* $\frac{1}{x(v) - x(u)}$. A *2D drawing* as in Fig. 1(a) is obtained from an intermediate drawing by transforming each point $p = (x(p), y(p))$ of the plane to $(x_{2D}(p), y_{2D}(p)) = (r(p) \cdot \cos(\alpha(p)), r(p) \cdot \sin(\alpha(p)))$, with the radius $r(p) = (\text{offset}_x + \max_{v \in V} (x(v))) - x(p) \cdot \delta_x$ and the angle $\alpha(p) = (y(p) - 1) \cdot \frac{2\pi}{k}$. The constant offset_x defines the minimum distance of a vertex to the center and δ_x the minimum distance of vertices on the same level. A *3D drawing* as in Fig. 1(c) uses the coordinates $(x_{3D}(p), y_{3D}(p), z_{3D}(p)) = (x(p) \cdot \delta_x, -r_k \cdot \sin(\alpha(p)), r_k \cdot \cos(\alpha(p)))$ where r_k is the radius of the cylinder. These equations transform straight lines of the intermediate drawing to spiral segments in the 2D or 3D drawings.

A drawing is (*cyclic level*) *plane* if the edges do not cross except on common endpoints. A cyclic k -level graph is (*cyclic level*) *planar* if such a drawing exists.

3 Layout Algorithm

In this section we describe our coordinate assignment phase for cyclic level graphs. We adapt the algorithm of Brandes and Köpf [3] and use their notation. Like them, we also assume that the crossing reduction has avoided type 2 conflicts. These can be avoided even for cyclic level graphs as shown recently [5].

The input to our algorithm is the output of the third phase and thus a proper ordered cyclic level graph. Note that dummy vertices were introduced after the leveling. Algorithm 1 consists of three basic steps: block building (lines 4–5), horizontal compaction (lines 6–12), and balancing (line 14) which reflect the steps in [3]. The first two steps are carried out four times (*runs*) for each combination of left/right with up/down alignment (line 2). The four results are merged by the balancing step. We describe the left top run only. The other three runs are realized by flipping the graph horizontally and/or vertically before and after (lines 3, 13) each run. The computed intermediate drawing can be transformed into the 2D or 3D drawing, where dummy vertices are replaced by edge bends.

In the cyclic case there may be unavoidable cyclic dependencies in the left-to-right ordering among vertically aligned paths. Thus, it is impossible to draw

Algorithm 1. cyclicCoordinateAssignment**Input:** $G = (V, E, \phi, <)$: An ordered and proper cyclic k -level graph**Output:** Coordinates $(x(v), y(v))$ for each $v \in V$ in the intermediate drawing $\mathcal{I}$

```

1  $\mathcal{P} \leftarrow \emptyset$ 
2 foreach  $(h, v) \in \{left, right\} \times \{up, down\}$  do
3    $G' \leftarrow \text{flip}(G, h, v)$  // according to current run
4    $H \leftarrow \text{buildCyclicBlockGraph}(G')$ 
5    $\text{splitLongBlocks}(H)$  // split long and closed blocks
6    $\mathcal{S} \leftarrow \text{computeSCCs}(H)$ 
7   foreach complex SCC  $S \in \mathcal{S}$  do
8      $S' \leftarrow \text{cutSCC}(S)$  // returns non-cyclic block graph
9      $\text{width}(S') \leftarrow \text{compact}(S')$  // using two topsorts
10     $\text{shear}(S', -(\text{wind}(S') \cdot k) / \text{width}(S'))$  // shear  $S'$  with given slope
11     $S \leftarrow S \setminus S' \cup S'$ 
12   $\text{compact}(\mathcal{S})$  // globally all SCCs
13   $\mathcal{P} \leftarrow \mathcal{P} \cup \text{flip}(\mathcal{S}, h, v)$ 
14  $\mathcal{I} \leftarrow \text{balance}(\mathcal{P})$  // balance four runs
15 return  $\mathcal{I}$ 

```

inner segments vertically, in general. We solve this problem by shearing the drawing of such a cycle s. t. all inner segments have the same slope.

3.1 Block Building

The block building phase is done in the same way as in [3]. We try to align vertices with its median adjacent vertices to blocks and remove all other segments level by level until we obtain a cyclic path graph and thus a cyclic block graph.

Definition 1. A cyclic path graph $H' = (V, E_{intra}, \phi, <)$ is an ordered and proper cyclic level graph with a plane embedding respecting the ordering $<$. Each vertex of H' has indegree and outdegree at most one. We call each connected component of H' a block and all edges $e \in E_{intra}$ intra block edges. A block B is closed if each vertex of B has indegree and outdegree one or open, otherwise. The height of B is defined as the number of intra block edges in B . The cyclic block graph $H = (V, E_{intra} \dot{\cup} E_{inter}, \phi)$ of H' is obtained by adding an edge $e \in E_{inter}$ from each vertex in H' to its consecutive right vertex on the same level (if there is one), which we call inter block edges.

To create such a graph we first mark outer segments involved in type 1 conflicts between two levels. Then, we traverse the lower level from left to right and try to align each vertex with one of its median predecessor vertices. First we try its upper left median, then its upper right median. An alignment is impossible if the segment is marked or if it would cross a segment already used for aligning. The current vertex becomes the top vertex of a new block if both alignments fail. All inner segments of an edge are aligned and thus lie in the same block.

As each block is drawn with constant slope, this ensures at most two bends per edge. E_{intra} is the set of all remaining edges. See Fig. 2 for an example. Vertices and intra block edges of the same block are framed. The inter block edges lie on the level lines. The dotted segments were removed in the block building phase.

The cyclic block graph can have closed blocks (with height k) and open blocks with height $\geq k$ (spirals) which shall be avoided to simplify the algorithm (line 5). In both cases we split such a block by removing outer segments until each resulting block has at most height $k - 1$. Such outer segments always exist, as no edge can span more than k levels. Therefore, the invariant of at most two bends per edge still holds. Note that an originally closed block will not be sheared like other blocks in Sect. 3.2 as it cannot be part of a cyclic dependency. See Fig. 2(b) for an example: The segment $(2, 4)$ was removed to open a closed block and the segment $(5, 7)$ was used to split a long block in two shorter ones. The result is a cyclic block graph with open blocks of height at most $k - 1$.

3.2 Horizontal Compaction

In this section we compact the cyclic block graph by arranging all blocks as close to each other as possible minimizing the width of the drawing. Not all blocks can be drawn vertically as there can be cyclic dependencies in the left-to-right ordering among blocks, which we call rings.

Definition 2. A block path P in a cyclic block graph $H = (V, E_{\text{intra}} \dot{\cup} E_{\text{inter}}, \phi)$ is a sequence of vertices $v_1, \dots, v_s \in V$ s. t. for each pair of consecutive vertices v_i and v_{i+1} , $1 \leq i < s$, $(v_i, v_{i+1}) \in E_{\text{intra}}$ or $(v_{i+1}, v_i) \in E_{\text{intra}}$ or $(v_i, v_{i+1}) \in E_{\text{inter}}$. It is simple if all vertices are mutually distinct. A block path is a ring R if $v_1 = v_s$. In a simple ring the vertices $v_1, \dots, v_{s-1}$ are mutually distinct. The width of R is the number of inter block edges in R . Let c_{down} and c_{up} be the number of intra block edges traversed in R in and against their direction, respectively. The number of windings of R is then defined as $\text{wind}(R) = (c_{\text{down}} - c_{\text{up}})/k$.

Informally, a ring is a cycle in the block graph where the direction of the inter block edges is preserved and the intra block edges are used in any direction. $\text{wind}(R)$ counts how often R wraps around the center. As each ring is an ordered sequence, we count windings along increasing and decreasing levels positive and negative, respectively. We consider the strongly connected components (SCCs) connected by block paths of the block graph separately. The *simple* SCCs consist of one block. All other *complex* SCCs contain rings. Figure 2(a) consists of three simple SCCs $((2, 4), (7, 9, 12)$ and $(13))$ and one complex SCC (the remaining two blocks) in which all simple rings R have $\text{width}(R) = 2$ and $\text{wind}(R) = 1$.

Lemma 1. For each ring R of a cyclic block graph G $\text{wind}(R) \neq 0$.

Proof. Assume for contradiction that there exists a ring R with $\text{wind}(R) = 0$ in the block graph of G . Unwrapping G several times, i. e., placing multiple copies of the intermediate drawing one below the other and merging first and last levels, leads to a block graph of a (non-cyclic) level graph H which contains R completely. Then, H has a cyclic dependency, which is a contradiction. $\square$

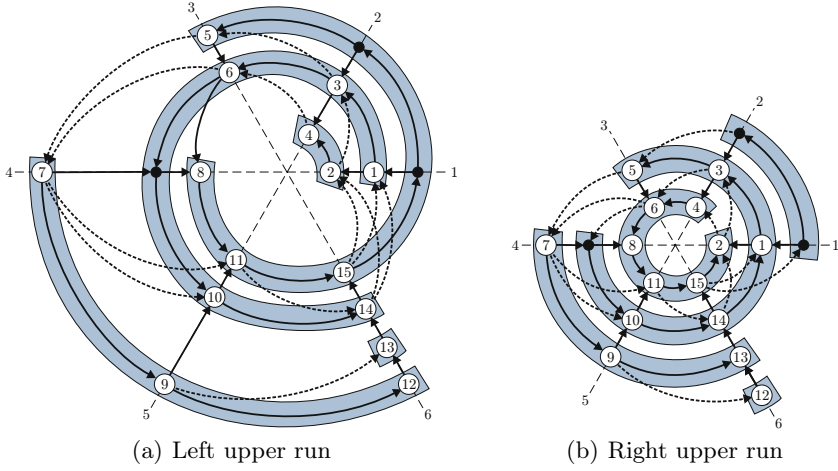


Fig. 2. Block graphs of Fig. 1(a) to (c) as 2D drawings

Lemma 2. For each simple ring R of a cyclic block graph $|\text{wind}(R)| \leq 1$.

Proof. Assume for contradiction that there is a simple ring R with $|\text{wind}(R)| > 1$. As R wraps around the center in the 2D drawing more than once, each drawing of R crosses itself. Each cyclic path graph is planar. Further, each drawing of it respecting its ordering can be extended to a planar drawing of its cyclic block graph by adding the inter block edges along the level lines. Since R is a subgraph of the cyclic block graph, this is a contradiction. $\square$

Theorem 1. Let $\mathcal{R}$ be the set of all simple rings of an SCC in a cyclic block graph. For each $R \in \mathcal{R}$ $\text{wind}(R) = 1$ or for each $R \in \mathcal{R}$ $\text{wind}(R) = -1$.

Proof. According to Lemmas 1 and 2 $|\text{wind}(R)| = 1$ holds for each ring $R \in \mathcal{R}$. Assume for contradiction that there exist two rings $R_1, R_2 \in \mathcal{R}$ with $\text{wind}(R_1) = 1$ and $\text{wind}(R_2) = -1$. Let v_1 and v_2 be vertices in R_1 and R_2 , respectively. Let S be a (not necessarily simple) ring through v_1 and v_2 , which always exists as v_1 and v_2 lie in the same SCC. Due to Lemma 1 $\text{wind}(S) \neq 0$. If $\text{wind}(S) > 0$, let T be a non-simple ring consisting of S and $\text{wind}(S)$ many copies of R_2 joined via v_2 . Otherwise, let T be a ring consisting of S and $-\text{wind}(S)$ many copies of R_1 joined via v_1 . In both cases $\text{wind}(T) = 0$, which contradicts Lemma 1. $\square$

Definition 3. Let S be a complex SCC of a cyclic block graph containing a simple ring R . We define $\text{wind}(S) = \text{wind}(R)$ and $\text{width}(S)$ as the maximum width of all simple rings in S .

Horizontal Compaction of a complex Strongly Connected Component.

It is not possible to draw all blocks of a complex SCC S straight-line and vertically. Therefore, we will draw all blocks of S with the same slope. The slope has to be chosen s. t. each ring, and thus the resulting curve, in S starts and ends at

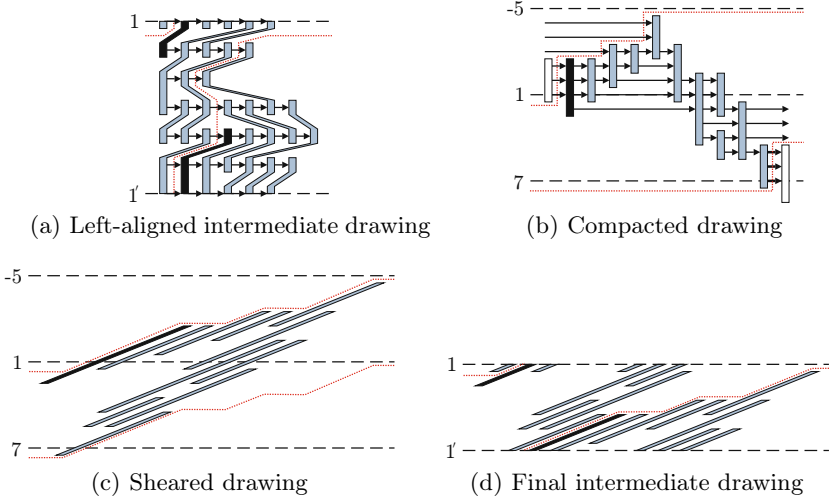


Fig. 3. Drawing of a complex SCC

the same coordinates. All rings in S have the same number of windings $\text{wind}(S)$, which is either 1 or -1 . Thus, each simple ring spans $\text{wind}(S) \cdot k$ levels. To draw one simple ring R we could use the slope $-(\text{wind}(R) \cdot k) / \text{width}(R)$, which would result in inter block edges of unit length. In order to draw all blocks in S with the same slope (line 10 in Algorithm 1) we must use the width of the widest simple ring of S and the slope $-(\text{wind}(S) \cdot k) / \text{width}(S)$. With this slope the widest ring will fit exactly and use unit length inter block edges. All narrower rings will have some unused horizontal space in the drawing and thus have inter block edges which are longer than one unit.

To use this slope we have to determine the width of the widest simple ring in S . The general problem of finding a longest cycle is $\mathcal{NP}$ -hard [4]. However, here it can be solved in linear time by cutting the SCC. Finding the length and compacting the layout is done simultaneously. To cut an SCC (line 8 in Algorithm 1), we start at an arbitrary block B and temporarily remove all incoming inter block edges of B . We then follow the outgoing inter block edge of the topmost vertex of B to the next block B' . We temporarily remove all incoming inter block edges of B' which are above the traversed incoming edge. We repeat this procedure until the topmost vertex of the current block has no outgoing inter block edge. Note that this happens before B is visited a second time, as otherwise the SCC would contain unreachable blocks. Thus, a block path P_r from B to a rightmost vertex in S is found. The same procedure is repeated from starting block B using the outgoing inter block edge of the lowest vertex to reach B' and deleting all incoming inter block edges below the traversed edge until a lowest vertex with no incoming inter block edge is found, which gives the block path P_l .

Combining P_r and P_l results in a y -monotone path P from a rightmost vertex through B to a leftmost vertex, using inter block edges in both directions and

preserving intra block edge directions. Due to the removed incoming inter block edges left of P all rings in S are cut exactly once. We then assign an arbitrary vertex $v \in V$ of B the new coordinate $y'(v) = \phi(v)$. In a traversal of the block graph we assign each vertex a y' -coordinate: Using an inter block edge we assign both end vertices the same y' -coordinate. Using an intra block edge in or against its direction we increase or decrease the y' -coordinate by 1, respectively, without using a modulo operation. The result is an acyclic block graph, which we compact (in contrast to [3]) in the following way (line 9 in Algorithm 1): We place each block which is a source in the acyclic block graph on an imaginary zero line, treat all other blocks in the topological order and move them as much to the left as possible preserving unit distance. Afterwards, we fix all sinks on their positions, treat all other blocks against the topological order and move them as much to the right as possible. For placing a block as close as possible to the already placed ones, we traverse its levels. After the compaction each block (and therefore each vertex v in S) has an assigned x' -coordinate. Let $e = (u, v)$ be a removed inter block edge. The width of the widest simple ring of S through e is then $x'(u) - x'(v) + 1$. Considering all removed inter block edges and computing the maximum value gives the width of the widest simple ring $\text{width}(S)$ in S .

See Fig. 3 for an example of an SCC S with $\text{wind}(S) = 1$ and $k = 6$ levels. Figure 3(a) shows a leftmost intermediate drawing with the black start block B . Its lowest vertex is already the leftmost vertex on its level. To reach a rightmost vertex four other blocks have to be visited. The dashed line cuts six inter block edges. Figure 3(b) shows the resulting compacted horizontal block graph using y' -coordinates. The widths of the widest rings through each of the six cutted edges are (from top to bottom): 5, 5, 7, 10, 10, 10. Thus, $\text{width}(S) = 10$. Shearing the drawing with slope $\frac{-1 \cdot 6}{10}$ results in Fig. 3(c). Using the modulo operation for the y -coordinates gives the final intermediate drawing in Fig. 3(d).

Theorem 2. *The intermediate drawing of S uses the coordinates $x(v) = x'(v) - (\text{width}(S)/(\text{wind}(S) \cdot k)) \cdot y'(v)$ and $y(v) = ((y'(v) - 1) \bmod k) + 1 = \phi(v)$ for each vertex v in S . In the drawing all intra block edges have slope $\text{slope}(S) = -(\text{wind}(S) \cdot k) / \text{width}(S)$. The ordering of vertices on the same level is the one given by the crossing reduction phase and these vertices have at least unit distance.*

Proof. In the compacted drawing of the open block graph all blocks are drawn vertically. Shearing the drawing results in unchanged y' -coordinates and new x -coordinates $x(v) = x'(v) + y'(v) / \text{slope}(S)$. Now all edges have slope $\text{slope}(S)$. Using the y -coordinates $y(v) = ((y'(v) - 1) \bmod k) + 1 = \phi(v)$ does not affect the slope of the edges. But now all vertices on the same level have the same y -coordinate again. Let u and v be two consecutive vertices on the same level. Let u be left of v according to the crossing reduction. If the inter block edge (u, v) was not cut before, then u and v have the same y' -coordinates in the compacted drawing and u is still the left neighbor of v with at least unit distance between them. This does not change in the sheared or final intermediate drawing. If (u, v) was cut, then $y'(v) = y'(u) - k \cdot \text{wind}(S)$. There exists a simple block path P from v to u as we are compacting an SCC. P cannot have been cut as otherwise P and (u, v) form a simple ring that would have been cut twice. The ring formed

by P and (u, v) is at most $\text{width}(S)$ wide and thus $x'(v) \geq x'(u) - (\text{width}(S) - 1)$. After the drawing is sheared, $x(v) \geq x(u) + 1$ holds. Therefore, u is still left of v and the two vertices are at least unit distance apart. As a result, all consecutive vertices and thus all vertices on the same level are separated by unit distance and are in the ordering given by the crossing reduction phase. $\square$

Horizontal Compaction. Our next step is to globally compact the set of compacted complex SCCs and simple SCCs (line 12 in Algorithm 1).

Lemma 3. *In a drawing that respects the order of the crossing reduction phase all vertices of an SCC on the same level are consecutive.*

Proof. Let u, v be two vertices of an SCC on the same level s. t. u is left of v . Note that there is a block path from v to u . Also, there is a horizontal path from u to v using inter block edges only. Therefore, all vertices between u and v lie on a ring containing u and v and thus belong to the same SCC. $\square$

This means that no SCCs can interleave. We interpret the SCCs as super vertices and perform a topological sorting on the resulting DAG. We then compact the SCCs as we compacted the blocks of a (non-cyclic) block graph before.

3.3 Balancing

In this phase (line 14 in Algorithm 1) the four results are balanced by computing one x -coordinate for each vertex out of the four x -coordinates computed by the four runs. The only difference to the algorithm of Brandes and Köpf [3] is that we do not use the average median of the four x -coordinates for each vertex, since this can induce additional bends in the cyclic case. The reason is that on lines with different slopes the median changes at crossings, i. e., it is a non-linear function. Hence, we use the average of all four x -coordinates for each vertex.

Proposition 1. *Using the average of the x -coordinates of the four runs for each vertex does not change the ordering of the vertices on a level and preserves at least unit distance. Additional bends can occur since the blocks of the four runs may differ. However, the invariant of at most two bends per edge e in the final drawing still holds, as the y -coordinates of the bends located at the topmost and lowest dummy vertex of e are identical in each drawing.*

Note that it is possible that some vertices in one run belong to a block of an SCC although they do not belong to an SCC or even one block in another run. Thus, balancing can lead to more different slopes than in each of the four runs alone. See Fig. 2 for two different block graphs of two runs of the same graph.

4 Algorithm Analysis

Theorem 3. *Let $G = (V, E, \phi, \prec)$ be a proper ordered cyclic k -level graph. The width of the intermediate drawing of G is $\mathcal{O}(|V|^2/k)$ and the area is $\mathcal{O}(|V|^2)$. For the 3D drawing the same bounds hold. The 2D drawing has a width and height of $\mathcal{O}(|V|^2/k)$ and thus an area of $\mathcal{O}(|V|^4/k^2)$.*

Proof. Let $\mathcal{S} = \{S_1, \dots, S_r\}$ be the set of all SCCs. Let N_i be the number of (dummy) vertices in S_i . The width of the compacted drawing of S_i is $\text{width}(S_i) \leq N_i$. Shearing the drawing of height at most N_i with slope $-\text{wind}(S) \cdot k / \text{width}(S_i)$ adds at most N_i^2/k to the width. Thus, the width of the drawing is in $\mathcal{O}(N_i^2/k)$. The width of the drawing of G is at most the sum of the widths of the drawings of all SCCs and thus in $\mathcal{O}(|V|^2/k)$. As the height is k , the area is in $\mathcal{O}(|V|^2)$.

The height and width of the 2D drawing is twice the width of the intermediate drawing and thus the area is $\mathcal{O}(|V|^4/k^2)$, however. $\square$

Note that the width of the drawing reduces to $\mathcal{O}(|V|)$ if there are no complex SCCs in the graph. This reduces the area of the intermediate and 3D drawings to $\mathcal{O}(|V| \cdot k)$ and of the 2D drawing to $\mathcal{O}(|V|^2)$. Complex SCCs can always be avoided by using special crossing reduction and block building algorithms [5].

Theorem 4. *The layout algorithm described in Algorithm 1 has a time complexity of $\mathcal{O}(|V| + |E|)$ for a proper ordered cyclic k -level graph $G = (V, E, \phi, <)$.*

5 Summary

We presented the first coordinate assignment algorithm for cyclic level graphs. Like the established algorithm by Brandes and Köpf, which is the de facto standard coordinate assignment method for hierarchic level graphs, we ensure to have at most two bends per edge and try to align long edges and center parents over their children. These are the major aesthetic criteria for such drawings. We implemented a prototype of our algorithm within the Gravisto toolkit [2].

References

1. Bachmaier, C., Brandenburg, F.J., Brunner, W., Lovász, G.: Cyclic leveling of directed graphs. In: Tollis, I., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 348–359. Springer, Heidelberg (2009)
2. Bachmaier, C., Brandenburg, F.J., Forster, M., Holleis, P., Raitner, M.: Gravisto: Graph visualization toolkit. In: Pach, J. (ed.) GD 2004. LNCS, vol. 3383, pp. 502–503. Springer, Heidelberg (2005), <http://gravisto.fim.uni-passau.de/>
3. Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 31–44. Springer, Heidelberg (2002)
4. Garey, M.R., Johnson, D.S.: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
5. Hübner, F.: A Global Approach on Crossing Minimization in Hierarchical and Cyclic Layouts of Leveled Graphs. Master's thesis, University of Passau (2009)
6. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
7. Michal, G. (ed.): Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology. Wiley, Chichester (1998)
8. Serafini, P., Ukovich, W.: A mathematical model for periodic scheduling problems. SIAM J. Discrete Math. 2(4), 550–581 (1989)
9. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst., Man, Cybern. 11(2), 109–125 (1981)

Crossing-Optimal Acyclic HP-Completion for Outerplanar *st*-Digraphs

Tamara Mchedlidze and Antonios Symvonis

Dept. of Mathematics, National Technical University of Athens, Athens, Greece
`{mchet,symvonis}@math.ntua.gr`

Abstract. Given an embedded planar acyclic digraph G , the *acyclic hamiltonian path completion with crossing minimization (Acyclic-HPCCM)* problem is to determine a *hamiltonian path completion set* of edges such that, when these edges are embedded on G , they create the smallest possible number of edge crossings and turn G to a hamiltonian acyclic digraph. In this paper, we present a linear time algorithm which solves the Acyclic-HPCCM problem on any outerplanar *st*-digraph G . The algorithm is based on properties of the optimal solution and an *st-polygon decomposition* of G . As a consequence of our result, we can obtain for the class of outerplanar *st*-digraphs upward topological 2-page book embeddings with minimum number of spine crossings.

1 Introduction

A *hamiltonian path* of G is a path that visits every vertex of G exactly once. Determining whether a graph has a hamiltonian path or circuit is NP-complete [3]. The problem remains NP-complete for cubic planar graphs [3], for maximal planar graphs [9] and for planar digraphs [3]. It can be trivially solved in polynomial time for planar acyclic digraphs.

Given a graph $G = (V, E)$, directed or undirected, a non-negative integer $k \leq |V|$ and two vertices $s, t \in V$, the *hamiltonian path completion (HPC)* problem asks whether there exists a superset E' containing E such that $|E' - E| \leq k$ and the graph $G' = (V, E')$ has a hamiltonian path from vertex s to vertex t . We refer to G' and to the set of edges $E' \setminus E$ as the *HP-completed graph* and the *HP-completion set* of graph G , respectively. We assume that all edges of a HP-completion set are part of the Hamiltonian path of G' , otherwise they can be removed. The hamiltonian path completion problem is NP-complete [2]. For acyclic digraphs the HPC problem is solved in polynomial time [4]. When G is a directed acyclic graph, we can insist on HP-completion sets which leave the HP-completed digraph also acyclic. We refer to this version of the problem as the *acyclic HP-completion problem (acyclic-HPC)*.

A *drawing* Γ of graph G maps every vertex v of G to a distinct point $p(v)$ on the plane and each edge $e = (u, v)$ of G to a simple open curve joining $p(u)$ with $p(v)$. A drawing in which every edge (u, v) is a simple open curve monotonically increasing in the vertical direction is an *upward drawing*. A drawing Γ of graph

G is *planar* if no two distinct edges intersect except at their end-vertices. Graph G is called *planar* if it admits a planar drawing Γ .

An embedding of a planar graph G is the equivalence class of planar drawings of G that define the same set of faces or, equivalently, of face boundaries. A planar graph together with the description of a set of faces F is called an *embedded planar graph*.

Let $G = (V, E)$ be an embedded planar graph, E' be a superset of edges containing E , and $\Gamma(G')$ be a drawing of $G' = (V, E')$. When the deletion from $\Gamma(G')$ of the edges in $E' - E$ induces the embedded planar graph G , we say that $\Gamma(G')$ *preserves the embedded planar graph G* .

Definition 1. *Given an embedded planar graph $G = (V, E)$, directed or undirected, a non-negative integer c , and two vertices $s, t \in V$, the **hamiltonian path completion with edge crossing minimization (HPCCM) problem** asks whether there exists a superset E' containing E and a drawing $\Gamma(G')$ of graph $G' = (V, E')$ such that (i) G' has a hamiltonian path from vertex s to vertex t , (ii) $\Gamma(G')$ has at most c edge crossings, and (iii) $\Gamma(G')$ preserves the embedded planar graph G .*

We refer to the version of the HPCCM problem where the input is an acyclic digraph and we are interested in HP-completion sets which leave the HP-completed digraph also acyclic as the *Acyclic-HPCCM problem*.

Over the set of all HP-completion sets for an embedded planar graph G , and over all of their different drawings that respect G , the one with a minimum number of edge-crossings is called a *crossing-optimal HP-completion set*.

In this paper, we present a linear time algorithm which solves the Acyclic-HPCCM problem for outerplanar *st*-digraphs. A planar graph G is *outerplanar* if there exist a drawing of G such that all of G 's vertices appear on the boundary of the same face (which is usually drawn as the external face). Let $G = (V, E)$ be a digraph. A vertex of G with in-degree equal to zero (0) is called a *source*, while, a vertex of G with out-degree equal to zero is called a *sink*. An *st-digraph* is an acyclic digraph with exactly one source and exactly one sink. Traditionally, the source and the sink of an *st-digraph* are denoted by s and t , respectively. An *st-digraph* which is planar (resp. outerplanar) and, in addition, it is embedded on the plane so that both of its source and sink appear on the boundary of its external face, is referred to as a **planar *st-digraph*** (resp. an **outerplanar *st-digraph***). It is known that a planar *st-digraph* admits a planar upward drawing [5,1]. In the rest of the paper, all *st-digraphs* will be drawn upward.

1.1 Our Results

The Acyclic-HPCCM problem was introduced by Mchedlidze and Symvonis in [8]. They provided a characterization under which a triangulated *st-digraph* is hamiltonian. For the class of planar *st-digraphs*, they established an equivalence between the acyclic-HPCCM problem and the problem of determining an upward 2-page topological book embedding with a minimal number of spine crossings.

For any outerplanar *triangulated st*-digraph G they reported a linear-time algorithm that solves the Acyclic-HPCCM problem *with at most one crossing per edge* of G [7].

In this paper, we derive a linear time algorithm that solves the Acyclic-HPCCM problem for any outerplanar *st*-digraph G . Our algorithm extends the results presented in [7] in two ways: (a) it does not require G to be triangulated, and (b) it takes into account all acyclic HP-completion sets (and not only those which cause at most 1 crossing per edge of G). The algorithm is based on properties of the optimal solution and a decomposition of graph G .

More specifically, we show that (i) for any *st*-polygon (i.e., an outerplanar *st*-digraph with no edge connecting its two opposite sides) there is always a crossing-optimal acyclic HP-completion set of size at most 2 (Section 2, Theorem 2), and, (ii) for any outerplanar *st*-digraph G there exists a crossing optimal acyclic HP-completion set which creates at most 2 crossings per edge of G (Section 4, Theorem 4). Based on these properties and the introduced *st*-polygon decomposition of an outerplanar *st*-digraph (Section 3), we derive a linear time algorithm that solves the Acyclic-HPCCM problem for outerplanar *st*-digraphs. Due to space constraints we present only the main results omitting the proofs of the lemmata and theorems; for the detailed version see [6].

2 Outerplanar *st*-Digraphs, Rhombuses and *st*-Polygons

Let $G = (V^l \cup V^r \cup \{s, t\}, E)$ be an outerplanar *st*-digraph, where s is its source, t is its sink and the vertices in V_l (resp. V_r) are located on the left (resp. right) part of the boundary of the external face. Let $V^l = \{v_1^l, \dots, v_k^l\}$ and $V^r = \{v_1^r, \dots, v_m^r\}$, where the subscripts indicate the order in which the vertices appear on the left (right) part of the external boundary. By convention, source and the sink are considered to lie on both the left and the right sides of the external boundary. Observe that each face of G is also an outerplanar *st*-digraph. We refer to an edge that has both of its end-vertices on the same side of G as an *one-sided* edge. All remaining edges are referred to as *two-sided* edges. The following lemma presents an essential property of an acyclic HP-completion set of an outerplanar *st*-digraph G .

Lemma 1. *The acyclic HP-completion set of an outerplanar st-digraph $G = (V^l \cup V^r \cup \{s, t\}, E)$ induces a hamiltonian path that visits the vertices of V_l (resp. V_r) in the order they appear on the left side (resp. right side) of G .*

The outerplanar *st*-digraph of Figure 1.a is called a *strong rhombus*. It has at least one vertex at each of its sides and it consists of exactly two faces that have edge (v_s, v_t) in common. The edge (v_s, v_t) of a rhombus is referred to as its *median* and is always drawn in the interior of its drawing. The outerplanar *st*-digraph resulting from the deletion of the median of a strong rhombus is referred to as a *weak rhombus* (Figure 1.b). We use the term *rhombus* to refer to either a strong or a weak rhombus.

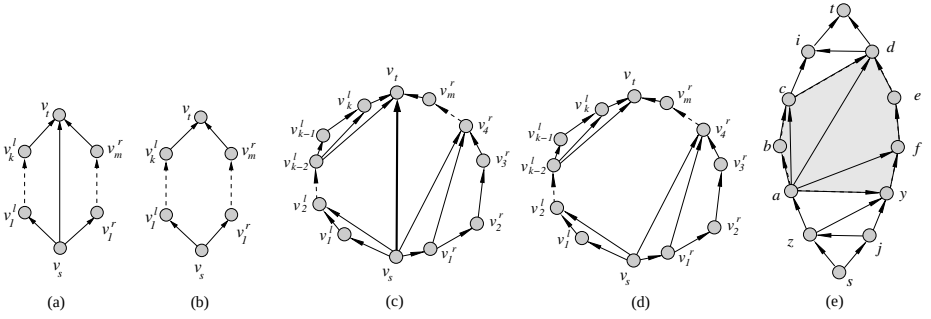


Fig. 1. (a) A strong rhombus. (b) A weak rhombus. (c) A strong st -polygon. (d) A weak st -polygon. (e) A maximal st -polygon.

The following theorem provides a characterization of st -digraphs that have a hamiltonian path. Its proof is a trivial extension of the proof given in [8] for triangulated planar st -digraphs.

Theorem 1. *Let G be a planar st -digraph. G has a hamiltonian path if and only if G does not contain any rhombus (strong or weak) as a subgraph.*

A *strong st -polygon* is an outerplanar st -digraph that contains edge (v_s, v_t) connecting its source v_s to its sink v_t (Figure 1.c). Edge (v_s, v_t) is referred to as its *median* and it always lies in the interior of its drawing. As a consequence, in a strong st -polygon no edge connects a vertex on its left side to a vertex on its right side. The outerplanar st -digraph that results from the deletion of the median of a strong st -polygon is referred to as a *weak st -polygon* (Figure 1.d). We use the term *st -polygon* to refer to both a strong and a weak st -polygon.

Consider an outerplanar st -digraph G and one of its embedded subgraphs G_p that is an st -polygon (strong or weak). G_p is called a *maximal st -polygon* if it cannot be extended (and still remain an st -polygon) by the addition of more vertices to its external boundary. In Figure 1.e, the st -polygon $G_{a,d}$ with vertices a (source), b , c , d (sink), e , and f on its boundary is not maximal since the subgraph $G'_{a,d}$ obtained by adding vertex y to it is still an st -polygon. However, the st -polygon $G'_{a,d}$ is maximal since the addition of either vertex i or z to it does not yield another st -polygon.

Observe that an st -polygon that is a subgraph of an outerplanar st -digraph G fully occupies a “strip” of it that is limited by two edges (one adjacent to its source and one to its sink), each having its endpoints at different sides of G . We refer to these two edges as the *limiting edges* of the st -polygon. Note that the limiting edges of an st -polygon that is an embedded subgraph of an outerplanar graph are sufficient to define it. In Figure 1.e, the maximal st -polygon with vertex a as its source and vertex d as its sink is limited by edges (a, y) and (c, d) . The next two lemmata describe properties of st -polygons.

Lemma 2. *An st -polygon contains exactly one rhombus.*

Lemma 3. *The maximal st -polygons contained in an outerplanar st -digraph G are mutually area-disjoint.*

The following lemmata are concerned with a crossing-optimal acyclic HP-completion set for a single st -polygon. They state that there exist crossing optimal acyclic HP-completion sets containing at most two edges and there proofs are based on the construction of a new HP-completion set.

Lemma 4. *Let $R = (V^l \cup V^r \cup \{s, t\}, E)$ be an st -polygon. Let P be an acyclic HP-completion set for R such that $|P| = 2\mu + 1$, $\mu \geq 1$. Then, there exists another acyclic HP-completion set P' for R such that $|P'| = 1$ and the edges of P' create at most as many crossings with the edges of R as the edges of P do. In addition, the hamiltonian paths induced by P and P' have in common their first and last edges.*

Lemma 5. *Let $R = (V^l \cup V^r \cup \{s, t\}, E)$ be an st -polygon. Let P be an acyclic HP-completion set for R such that $|P| = 2\mu$, $\mu \geq 1$. Then, there exists another acyclic HP-completion set P' for R such that $|P'| = 2$ and the edges of P' create at most as many crossings with the edges of R as the edges of P do. In addition, the hamiltonian paths induced by P and P' have in common their first and last edges.*

By combining Lemma 4 with Lemma 5, we get:

Theorem 2. *Any st -polygon has a crossing-optimal acyclic HP-completion set of size at most 2.*

3 st -Polygon Decomposition of Outerplanar st -Digraphs

The following lemmata state that we can test effectively whether an edge is the median of an st -polygon and whether a face is a weak-rhombus. Their proofs are supported by trivial data structures (i.e., for each vertex, a list of neighbors in circular order).

Lemma 6. *Assume an n vertex outerplanar st -digraph $G = (V^l \cup V^r \cup \{s, t\}, E)$ and an arbitrary edge $e = (s', t') \in E$. If $O(n)$ time is available for the preprocessing of G , we can decide in $O(1)$ time whether e is a median edge of some strong st -polygon. Moreover, the two vertices (in addition to s' and t') that define a maximal strong st -polygon that has edge e as its median can be also computed in $O(1)$ time.*

Lemma 7. *Assume an n vertex outerplanar st -digraph $G = (V^l \cup V^r \cup \{s, t\}, E)$ and a face f with source s' and sink v' . If $O(n)$ time is available for the preprocessing of G , we can decide in $O(1)$ time whether f is a weak rhombus. Moreover, the two vertices (in addition to s' and v') that define a maximal weak st -polygon that contains f can be also computed in $O(1)$ time.*

Denote by $\mathcal{R}(G)$ the set of all maximal st -polygons of an outerplanar st -digraph G , as identified by Lemmata 6 and 7. Observe that not every vertex of G belongs to one of its maximal st -polygons. We refer to the vertices of G that are not part of any maximal st -polygon as *free vertices* and we denote them by $\mathcal{F}(G)$. For example, vertices s , i , j , z and t in the st -digraph of Figure 1.e are free vertices. Also observe that an ordering can be imposed on the maximal st -polygons of an outerplanar st -digraph G based on the ordering of the area disjoint strips occupied by each st -polygon. The vertices which do not belong to some st -polygon are located in the area between the strips occupied by consecutive st -polygons.

The next lemma states that the free vertices, the sources and the sinks of st -polygon are pairwise connected by directed paths.

Lemma 8. *Assume an outerplanar st -digraph G . Let R_1 and R_2 be two of G 's consecutive maximal st -polygons and let $V_f \subset \mathcal{F}(G)$ be the set of free vertices lying between R_1 and R_2 . Then, the following statements are satisfied:*

- a) *For any pair of vertices $u, v \in V_f$ there is either a path from u to v or from v to u .*
- b) *For any vertex $v \in V_f$ there is a path from the sink of R_1 to v and from v to the source of R_2 .*
- c) *If $V_f = \emptyset$, then there is a path from source of R_1 to the source of R_2 .*

We refer to the source vertex s_i of each maximal st -polygon $R_i \in \mathcal{R}(G)$, $1 \leq i \leq |\mathcal{R}(G)|$ as the *representative* of R_i and we denote it by $r(R_i)$. We also define the representative of a free vertex $v \in \mathcal{F}(G)$ to be v itself, i.e. $r(v) = v$. For any two distinct elements $x, y \in \mathcal{R}(G) \cup \mathcal{F}(G)$, we define the relation $\angle_p$ as follows: $x \angle_p y$ iff there exists a path from $r(x)$ to $r(y)$.

Lemma 9. *Let G be an n node outerplanar st -digraph. Then, relation $\angle_p$ defines a total order on the elements $\mathcal{R}(G) \cup \mathcal{F}(G)$. Moreover, this total order can be computed in $O(n)$ time.*

Definition 2. *Given an outerplanar st -digraph G , the st -polygon decomposition $\mathcal{D}(G)$ of G is defined to be the total order of its maximal st -polygons and its free vertices induced by relation $\angle_p$.*

The following theorem follows directly from Lemma 6, Lemma 7 and Lemma 9.

Theorem 3. *An st -polygon decomposition of an n node outerplanar st -digraph G can be computed in $O(n)$ time.*

4 Crossing-Optimal Acyclic HP-Completion Set

In this section, we present some properties of a crossing optimal acyclic HP-completion for an outerplanar st -digraph that will be taken into account in our algorithm. Assume an outerplanar st -digraph $G = (V^l \cup V^r \cup \{s, t\}, E)$ and its st -polygon decomposition $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$. By G_i we denote the graph induced by the vertices of elements $o_1, \dots, o_i$, $i \leq \lambda$. The results of this section are summarized in the following Theorem.

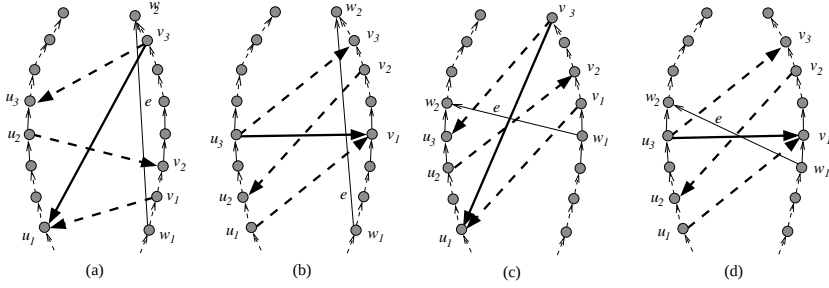


Fig. 2. Configurations of crossing edges used in the proof of Property 1

Theorem 4. Let $G = (V^l \cup V^r \cup \{s, t\}, E)$ be an outerplanar st -digraph and let $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$ be its st -polygon decomposition. Then, there exists a crossing optimal acyclic HP-completion set P_{opt} for G such that it satisfies the following properties:

- a) Each edge of E is crossed by at most two edges of P_{opt} .
- b) The upper limiting edge e_i of any maximal st -polygon o_i , $i \leq \lambda$, is crossed by at most one edge of P_{opt} . Moreover, the edge crossing e_i , if any, enters G_i .

The proof of Theorem 4 follows immediately from the following properties.

Property 1. Let $G = (V^l \cup V^r \cup \{s, t\}, E)$ be an outerplanar st -digraph. Then, no edge of E is crossed by more than 2 edges of a crossing-optimal acyclic HP-completion set for G .

Proof (sketch). For the sake of contradiction, assume that P_{opt} is a crossing-optimal acyclic HP-completion set for G , the edges of which cross some edge $e = (w_1, w_2)$ of G three times. Suppose the edges crossing e are e_1, e_2, e_3 . On Figures 2.a-d the edges $e_1 = (v_1, u_1)$, $e_2 = (u_2, v_2)$ and $e_3 = (v_3, u_3)$ are drawn by bold dashed line.

The idea of the proof is to show that we can obtain an acyclic HP-completion set for G that induces a smaller number of crossings than P_{opt} , a clear contradiction. We assume that all edges of P_{opt} participate in the hamiltonian path of G ; otherwise they can be discarded.

We distinguish between two cases based on whether edge e is a one-sided or a two-sided edge and for each of these cases we consider the both directions of e_1 , from the right to the left and from the left to the right. In all of these cases we substitute the edges e_1, e_2, e_3 of the HP-completion set by a single edge decreasing the total number of crossing created by the HP-completion set (see the bold solid edge on Figures 2.a-d for each case respectively). $\square$

Property 2. Let $G = (V^l \cup V^r \cup \{s, t\}, E)$ be an outerplanar st -digraph and let $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$ be its st -polygon decomposition. Then, there exists a crossing optimal acyclic HP-completion set for G such that, for every maximal

st-polygon $o_i \in \mathcal{D}(G)$, $i \leq \lambda$, the HP-completion set contains at most one edge that crosses the upper limiting edge of o_i and, moreover, this edge enters G_i . $\square$

5 The Algorithm

The algorithm for obtaining a crossing-optimal acyclic HP-completion set for an outerplanar *st*-digraph G is a dynamic programming algorithm based on the *st*-polygon decomposition $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$ of G . The following lemmata allow us to compute a crossing-optimal acyclic HP-completion set for an *st*-polygon and to obtain a crossing-optimal acyclic HP-completion set for G_{i+1} by combining an optimal solution for G_i with an optimal solution for o_{i+1} .

Assume an outerplanar *st*-digraph G . We denote by $S(G)$ the hamiltonian path on the HP-extended digraph of G that results when a crossing-optimal HP-completion set is added to G . Note that if we are only given $S(G)$ we can infer the size of the HP-completion set and the number of edge crossings. Denote by $c(G)$ the number of edge crossings caused by the HP-completion set inferred by $S(G)$. If we are restricted to Hamiltonian paths that enter the sink of G from a vertex on the left (resp. right) side of G , then we denote the corresponding size of HP-completion set as $c(G, L)$ (resp. $c(G, R)$). Obviously, $c(G) = \min\{c(G, L), c(G, R)\}$. Moreover, the notation can be extended to denote by $c^i(G, L)$ ($c^i(G, R)$) the corresponding minimum number of crossings over all HP-completion sets that contain exactly i edges. By Theorem 2, we know that the size of a crossing-optimal acyclic HP-completion set for an *st*-polygon is at most 2. This notation that restricts the size of the HP-completion set will be used only for *st*-polygons and thus, only the terms $c^1(G, L)$, $c^1(G, R)$, $c^2(G, L)$ and $c^2(G, R)$ will be utilized.

We use the operator $\oplus$ to indicate the concatenation of two paths. By convention, the hamiltonian path of a single vertex is the vertex itself.

The next lemma (follows from the Lemmata 4 and 5) states that it is sufficient to examine all HP-completion sets with one or two edges in order to find a crossing-optimal acyclic HP-completion set for an *st*-polygon.

Lemma 10. *Assume an n vertex *st*-polygon $o = (V^l \cup V^r \cup \{s, t\}, E)$. A crossing-optimal acyclic HP-completion set for o and the corresponding number of crossings can be computed in $O(n)$ time.*

Let $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$ be the *st*-polygon decomposition of G , where element o_i , $1 \leq i \leq \lambda$ is either an *st*-polygon or a free vertex. Recall that, we denote by G_i , $1 \leq i \leq \lambda$ the graph induced by the vertices of elements $o_1, \dots, o_i$. Graph G_i is also an outerplanar *st*-digraph. The same holds for the subgraph of G that is induced by any number of consecutive elements of $\mathcal{D}(G)$.

The next two lemmata describe how to combine the optimal solutions for consecutive *st*-polygons into an optimal solution for the whole outerplanar *st*-digraph.

Algorithm 1. ACYCLIC-HPC-CM(G)

input : An Outerplanar st -digraph $G(V^l \cup V^r \cup \{s, t\}, E)$.

output : The minimum number of edge crossing $c(G)$ resulting from the addition of a crossing-optimal acyclic HP-completion set to graph G .

1. Compute the st -polygon decomposition $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$ of G ;
 2. For each element $o_i \in \mathcal{D}(G)$, $1 \leq i \leq \lambda$,
compute $c^1(o_i, L)$, $c^1(o_i, R)$ and $c^2(o_i, L)$, $c^2(o_i, R)$:
 - if** o_i is a free vertex, **then** $c^1(o_i, L) = c^1(o_i, R) = c^2(o_i, L) = c^2(o_i, R) = 0$.
 - if** o_i is an st -polygon, **then** $c^1(o_i, L)$, $c^1(o_i, R)$, $c^2(o_i, L)$, $c^2(o_i, R)$ are computed

based on Lemma 10.
 3. **if** o_1 is a free vertex, **then** $c(G_1, L) = c(G_1, R) = 0$;
else $c(G_1, L) = \min\{c^1(o_1, L), c^2(o_1, L)\}$ and
 $c(G_1, R) = \min\{c^1(o_1, R), c^2(o_1, R)\}$;
 4. For $i = 1 \dots \lambda - 1$, compute $c(G_{i+1}, L)$ and $c(G_{i+1}, R)$ as follows:
 - if** o_{i+1} is a free vertex, **then**
 $c(G_{i+1}, L) = c(G_{i+1}, R) = \min\{c(G_i, L), c(G_i, R)\}$;
 - else-if** o_{i+1} is an st -polygon sharing **at most** one vertex with G_i , **then**
 $c(G_{i+1}, L) = \min\{c(G_i, L), c(G_i, R)\} + \min\{c^1(o_{i+1}, L), c^2(o_{i+1}, L)\}$;
 $c(G_{i+1}, R) = \min\{c(G_i, L), c(G_i, R)\} + \min\{c^1(o_{i+1}, R), c^2(o_{i+1}, R)\}$;
 - else** $\{ o_{i+1}$ is an st -polygon sharing **exactly** two vertices with $G_i\}$,
if $t_i \in V^l$, **then**
 $c(G_{i+1}, L) = \min\{c(G_i, L) + c^1(o_{i+1}, L) + 1, c(G_i, R) + c^1(o_{i+1}, L),$
 $c(G_i, L) + c^2(o_{i+1}, L), c(G_i, R) + c^2(o_{i+1}, L)\}$
 $c(G_{i+1}, R) = \min\{c(G_i, L) + c^1(o_{i+1}, R), c(G_i, R) + c^1(o_{i+1}, R),$
 $c(G_i, L) + c^2(o_{i+1}, R) + 1, c(G_i, R) + c^2(o_{i+1}, R)\}$
else $\{ t_i \in V^r \}$
 $c(G_{i+1}, L) = \min\{c(G_i, L) + c^1(o_{i+1}, L), c(G_i, R) + c^1(o_{i+1}, L),$
 $c(G_i, L) + c^2(o_{i+1}, L), c(G_i, R) + c^2(o_{i+1}, L) + 1\}$
 $c(G_{i+1}, R) = \min\{c(G_i, L) + c^1(o_{i+1}, R), c(G_i, R) + c^1(o_{i+1}, R) + 1,$
 $c(G_i, L) + c^2(o_{i+1}, R), c(G_i, R) + c^2(o_{i+1}, R)\}$
 5. **return** $c(G) = \min\{c(G_\lambda, L), c(G_\lambda, R)\}$
-

Lemma 11. Assume an outerplanar st -digraph G and let $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$ be its st -polygon decomposition. Consider any two consecutive elements o_i and o_{i+1} of $\mathcal{D}(G)$ that share at most one vertex. Then, the following statements hold:

- a) $S(G_{i+1}) = S(G_i) \oplus S(o_{i+1})$, and
- b) $c(G_{i+1}) = c(G_i) + c(o_{i+1})$.

Lemma 12. Assume an outerplanar st -digraph G and let $\mathcal{D}(G) = \{o_1, \dots, o_\lambda\}$ be its st -polygon decomposition. Consider any two consecutive elements o_i and o_{i+1} of $\mathcal{D}(G)$ that share an edge. Then, the following statements hold:

1. $t_i \in V^l \Rightarrow c(G_{i+1}, L) = \min\{ c(G_i, L) + c^1(o_{i+1}, L) + 1, c(G_i, R) + c^1(o_{i+1}, L),$
 $c(G_i, L) + c^2(o_{i+1}, L), c(G_i, R) + c^2(o_{i+1}, L)\}$

$$2. t_i \in V^l \Rightarrow c(G_{i+1}, R) = \min\{ c(G_i, L) + c^1(o_{i+1}, R), c(G_i, R) + c^1(o_{i+1}, R), \\ c(G_i, L) + c^2(o_{i+1}, R) + 1, c(G_i, R) + c^2(o_{i+1}, R) \}$$

$$3. t_i \in V^r \Rightarrow c(G_{i+1}, L) = \min\{ c(G_i, L) + c^1(o_{i+1}, L), c(G_i, R) + c^1(o_{i+1}, L), \\ c(G_i, L) + c^2(o_{i+1}, L), c(G_i, R) + c^2(o_{i+1}, L) + 1 \}$$

$$4. t_i \in V^r \Rightarrow c(G_{i+1}, R) = \min\{ c(G_i, L) + c^1(o_{i+1}, R), c(G_i, R) + c^1(o_{i+1}, R) + 1, \\ c(G_i, L) + c^2(o_{i+1}, R), c(G_i, R) + c^2(o_{i+1}, R) \}$$

Algorithm 1 is a dynamic programming algorithm, based on Lemmata 11 and 12, which computes the minimum number of edge crossings $c(G)$ resulting from the addition of a crossing-optimal HP-completion set to an outerplanar st -digraph G . The algorithm can be easily extended to also compute the corresponding hamiltonian path $S(G)$.

Theorem 5. *Given an n vertex outerplanar st -digraph G , a crossing-optimal HP-completion set for G and the corresponding number of edge-crossings can be computed in $O(n)$ time.*

We note that, from Theorem 2 in [8] and our presented algorithm, the following theorem is implied.

Theorem 6. *Given an n vertex outerplanar st -digraph G , an upward 2-page topological book embedding for G with minimum number of spine crossings can be computed in $O(n)$ time.*

References

1. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. *Theor. Comput. Sci.* 61(2-3), 175–198 (1988)
2. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., New York (1979)
3. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified np-complete problems. In: *STOC 1974: Proceedings of the sixth annual ACM symposium on Theory of computing*, pp. 47–63. ACM Press, New York (1974)
4. Karejan, Z.A., Mosesjan, K.M.: The hamiltonian completion number of a digraph (in Russian). *Akad. Nauk Armyan. SSR Dokl.* 70(2-3), 129–132 (1980)
5. Kelly, D.: Fundamentals of planar ordered sets. *Discrete Math.* 63, 197–216 (1987)
6. Mchedlidze, T., Symvonis, A.: Crossing-optimal acyclic hp-completion for outerplanar st -digraphs, arXiv:0904.2129, <http://arxiv.org/abs/0904.2129>
7. Mchedlidze, T., Symvonis, A.: Optimal acyclic hamiltonian path completion for outerplanar triangulated st -digraphs (with application to upward topological book embeddings), arXiv:0807.2330, <http://arxiv.org/abs/0807.2330>
8. Mchedlidze, T., Symvonis, A.: Crossing-optimal acyclic hamiltonian path completion and its application to upward topological book embeddings. In: Das, S., Uehara, R. (eds.) *WALCOM 2009. LNCS*, vol. 5431, pp. 250–261. Springer, Heidelberg (2009)
9. Wigderson, A.: The complexity of the hamiltonian circuit problem for maximal planar graphs. Technical Report TR-298, Princeton University, EECS Department (1982)

Edge-Intersection Graphs of k -Bend Paths in Grids

Therese Biedl¹ and Michal Stern²

¹ David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

`biedl@uwaterloo.ca`

² Caesarea Rothschild Institute, University of Haifa, Israel, and The Academic College of Tel-Aviv - Yaffo, Israel

`stern@mta.ac.il`

Abstract. Edge-intersection graphs of paths in grids are graphs that can be represented with vertices as paths in grids and edges between the vertices of the graph exist whenever two grid paths share a grid edge. This type of graphs is motivated by applications in conflict resolution of paths in grid networks.

In this paper, we continue the study of edge-intersection graphs of paths in a grid, which was initiated by Golumbic, Lipshteyn and Stern. We show that for any k , if the number of bends in each path is restricted to be at most k , then not all graphs can be represented. Then we study some graph classes that can be represented with k -bend paths, for small k . We show that every planar graph has a representation with 5-bend paths, every outerplanar graph has a representation with 3-bend paths, and every bipartite planar graph has a representation with 2-bend paths.

1 Introduction

Presume you have a network and need to route calls in it. Since calls interfere with each other, you need to route calls such that no two connections share a link. This transforms into a colouring problem in an edge-intersection graph of paths. The network is the host graph, and each call becomes a path in the host graph; calls share a link if and only if the paths share an edge of the host graph. The edge-intersection graph defined by this has a vertex for every path, and vertices are adjacent if and only if the paths have overlapping edges; the goal is hence to colour this edge-intersection graph.

Edge-intersection graphs of paths in a network have been studied almost exclusively for networks that are trees; this graph class is known as *EPT graphs* and was introduced over 20 years ago, though research is still ongoing (see [7] and the references therein.)

Very recently, Golumbic, Lipshteyn and Stern generalized the framework by allowing networks that are grids, rather than trees [6]. Thus, they define *EPG graphs* to be the edge-intersection graphs of paths in a grid. They showed that every graph is an EPG graph, and then restricted the graph class by limiting

the number of *bends* for the paths, i.e., the number of times that the direction of the path switches from horizontal to vertical. Focusing on the case of single bends, they proved a number of existence and impossibility results.

In this paper, we continue this work, and study graphs that can be represented with few (but more than 1) bends in each path. We first show that for any given k , there exist graphs that are not k -bend EPG graphs. We then study planar graphs and subclasses thereof, and give upper and lower bounds on the number of bends required for planar graphs, outerplanar graphs and planar bipartite graphs. We also study some other graph classes, such as line graphs, graphs of bounded path-width, and graphs that have edge orientations with bounded indegrees.

2 Definitions

We assume familiarity with graph theory notation, see for example Golumbic's book [5]. In this paper, *grid* is used to denote the 2-way infinite orthogonal grid, i.e., the vertices are all points in 2D with integer coordinates, and grid points are adjacent if they have distance 1. A (*grid*) *path* is a path in the grid. A *bend* is a place where a grid path changes direction. A (*grid*) *segment* is a grid path without any bends. Two grid paths (*edge*-)*intersect* if they share at least one grid edge, otherwise they are (*edge*-)*disjoint*. Note that edge-disjoint grid paths may still share a vertex of the grid. As we will never consider any other kind of intersection, we omit 'edge-' from now on. We will also omit "grid" whenever this does not lead to confusion.

An *EPG representation* of a graph $G = (V, E)$ is an assignment of grid paths to vertices of G such that (v, w) is an edge if and only if the paths assigned to v and w intersect. We call it a k -*bend EPG representation* if every grid path representing a vertex has at most k bends. A graph is called a k -*bend EPG graph* if it has a k -bend EPG representation. (These graphs were called B_k -EPG graphs in [6].) In what follows, we often identify the graph-theoretic concept (such as vertex) with the geometric object that represents it (the grid path).

An EPG representation is said to have grid-size $w \times h$ if there are w columns and h rows that contain a bend or endpoint of a grid path representing a vertex.

3 k -Bend EPG Graphs

In this section, we will show that the complete bipartite graph $K_{(k+3)/2, N}$ is not a k -bend EPG graph, for some $N \in O(k^4)$. As we learned only recently, a similar result, without asymptotic bounds on N , was discovered independently [1].

Theorem 1. $K_{(k+3)/2, N}$ is not a k -bend EPG graph for some $N \in O(k^4)$.

Proof. Set $\ell = (k + 3)/2$ and $N = 2^{\binom{\ell(k+1)}{2}} + 1 \in O(k^4)$. Let $V = A \cup B$ be the vertex partition of $K_{\ell, N}$, with $|B| = N$. Assume for contradiction that $K_{\ell, N}$ has a k -bend EPG representation, and let S be the grid segments of the paths representing vertices in A ; S contains at most $\ell(k + 1)$ many segments.

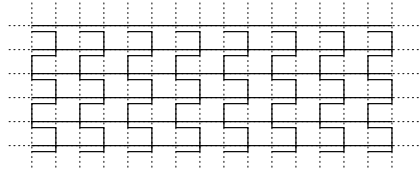


Fig. 1. An EPG representation of a complete bipartite graph in a 16×6 -grid

Every path representing a vertex in B must intersect ℓ of the segments in S , and no two of these paths can intersect such segments at the same place since the vertices in B are an independent set.

Consider two segments s_1 and s_2 in S , and assume that P is a path that intersects both and has at most one bend inbetween. If s_1 and s_2 have the same orientation (horizontal or vertical), then P cannot have a bend inbetween them, and hence must contain an endpoint of each of s_1 and s_2 . If s_1 and s_2 have different orientation, then P must contain a bend between them, and it necessarily must be on the point where the two lines through s_1 and s_2 intersect. So in the first case there can be at most one, and in the second case at most two disjoint paths that go from s_1 to s_2 with at most one bend inbetween.

Considering now all $\binom{\ell(k+1)}{2}$ possible pairs of grid segments. So at most $2\binom{\ell(k+1)}{2}$ grid paths can contain two of them consecutively with at most one bend inbetween. By choice of N , there is at least one vertex in B for which the representing path hence contains at least two bends between any two intersections with grid segments in S . Since the path intersects ℓ grid segments, it has at least $2\ell - 2 = k + 1$ bends total, a contradiction. $\square$

We conjecture that the value of N in the above theorem is too large, and that already $K_{\ell, N}$, for some $\ell \in \theta(k)$ and $N \in \theta(k^2)$ is not a k -bend EPG graph.

Theorem 2. *Every bipartite graph $G = (A \cup B, E)$ has a $(2|A| - 2)$ -bend EPG representation with grid-size $|A| \times 2|B|$.*

Proof. Fig. 1 illustrates the construction for the complete bipartite graph $K_{a,b}$. (Our construction is essentially the same as the one by Golumbic et al. [6], except that one bend can be saved since the graph is bipartite.) Represent each of the a vertices as a horizontal line, say at y -coordinates $1, 2, \dots, a$. Define a “vertical zig-zag” to be the path $(1, 1) - (2, 1) - (2, 2) - (1, 2) - \dots - (1, 2i-1) - (2, 2i-1) - (2, 2i) - (1, 2i) - \dots$ that ends with horizontal segment $(1, b) - (2, b)$ or $(2, b) - (1, b)$. Represent each of the b vertices by such a zig-zag, translated horizontally as to make them vertex-disjoint. (For the complete bipartite graph, they could in fact be placed even closer so that they share vertices; this would reduce the width to $|B| + 1$.) One easily verifies that this is an EPG representation of $K_{a,b}$ with the desired properties. For an arbitrary bipartite graph, we can obtain an EPG representation similarly, by omitting a “zig” or “zag” at any pair of vertices where no edge exists. $\square$

Notice in particular that $K_{(k+3)/2, N}$ is *not* a k -bend EPG graph for sufficiently large N by Theorem 1, but it *is* a $(k+1)$ -bend EPG graph by the Theorem 2. So the bound on the number of bends in Theorem 2 is tight if $|B|$ is large (but can be improved if $|B|$ is small [1].)

4 Planar Graphs

A *planar graph* is a graph that can be drawn without crossing in the 2-dimensional plane. Much is known about planar graphs (see e.g. [8]). We will study here bounds on the number of bends needed in EPG representations, both for planar graphs and for some of their subclasses.

4.1 Planar Graphs Are 5-Bend EPG Graphs

We show that every planar graph has a 5-bend EPG representation. Moreover, the layout of the paths for this is such that the paths only touch; they do not cross (in some sense, this is therefore a planar drawing.)

Theorem 3. *Every planar graph has a 5-bend EPG representation in an $n \times 2n$ -grid.*

Proof. We obtain this result by using a representation of planar graphs known as *T-drawing*. In such a drawing, every vertex is represented by an upside-down T (a horizontal bar with a vertical bar attached at the top), different Ts are disjoint except possibly at the endpoints of bars, and two vertices are adjacent if and only if their T's *touch*, i.e., have points in common (such points are necessarily at an endpoint of a bar.) Clearly every graph that has a T-drawing must be planar. De Fraysseix, de 2 and Rosenstiehl proved that every planar graph has such a T-drawing [4].

Following their proof, one can see that we can even impose the following restriction on the T-drawing:

- No two vertical bars have the same x -coordinate,
- No upper end of a vertical bar coincides with the end of a horizontal bar,
- The endpoints of bars are in an $n \times n$ -grid.

Now we convert the T-drawing to 5-bend paths by, roughly speaking, tracing along the Ts and adding handles at the end. More precisely, if a T has horizontal bar $[x_0, x_2] \times y_1$ and vertical bar $x_1 \times [y_1, y_2]$, then replace it by the path

$$(x_0, y_1 + \frac{1}{2}), (x_0, y_1), (x_2, y_1), (x_2, y_1 + \frac{1}{2}), (x_1, y_1 + \frac{1}{2}), (x_1, y_2), (x_1 + 1, y_2).$$

See Fig. 2. By the properties imposed onto the T-drawing, one easily verifies that this is an EPG representation of the graph with the desired properties after doubling y -coordinates. $\square$

On the other hand, we can easily see that some planar graphs are not 1-bend EPG graphs, by applying Theorem 1 to the planar graph $K_{2, N}$, for N sufficiently large. We strongly suspect that some planar graphs are not 2-bend EPG graphs either (and possibly not even 4-bend EPG graphs), but this remains open.

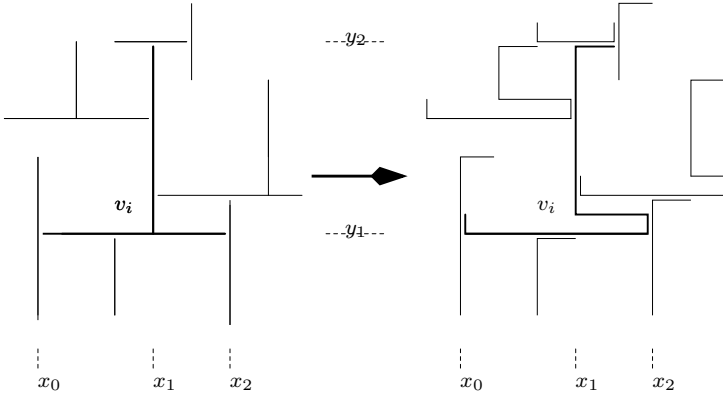


Fig. 2. Converting a T-drawing to a 5-bend EPG representation

4.2 Planar Bipartite Graphs

Now consider graphs that are both planar and bipartite. We have already seen in Theorem 1 that these are not always 1-bend EPG graphs, since $K_{2,N}$ is planar and bipartite.

Theorem 4. *Every planar bipartite graph $G = (A \cup B, E)$ has a 2-bend EPG representation of size $(|A| + 1) \times (|B| + 1)$.*

Proof. It is known [3] that every planar bipartite graph can be represented by touching horizontal and vertical line segments, i.e., vertices are assigned to disjoint open segments such that (v, w) is an edge if and only if the closure of the segments of v and w intersect. Furthermore, one can achieve that these segments are in an $|A| \times |B|$ -grid and no two segments are on a horizontal or vertical line.

Let s_h and s_v be a horizontal and vertical segment that touch at point (x, y) . For at least one of them, this point must be an endpoint. If only s_v ends at (x, y) , then add segment $[x, x + 1] \times y$ to it. If only s_h ends at (x, y) , then add segment $x \times [y, y + 1]$ to the grid path replacing s_h . If they both end at (x, y) , then add segment $x \times [y, y + 1]$ to both (if not in s_v already.) Put differently, we replace a vertical segment by a C and a horizontal segment by a U, except when both segments end at the same point. We obtain a 2-bend EPG representation with the desired properties. $\square$

4.3 Outer-Planar Graphs

An *outer-planar graph* is a planar graph that can be drawn so that all vertices are on the outer-face. Since $K_{2,N}$ is not an outer-planar graph (for $N \geq 3$), one could suspect that these are all 1-bend EPG graphs. This is not the case.

Theorem 5. *There exists an outer-planar graph that is not a 1-bend EPG graph.*

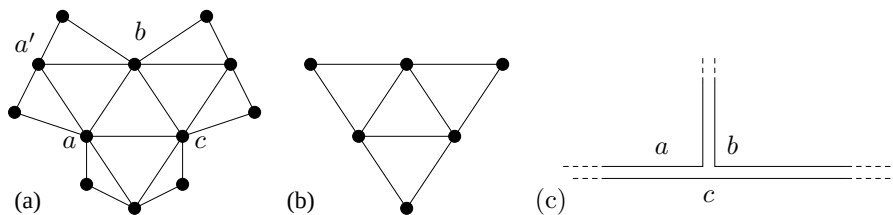


Fig. 3. (a) An outer-planar graph that is not a 1-bend EPG graph. (b) The 3-sun. (c) A claw-clique.

Proof. The graph G shown in Fig. 3(a) is an outer-planar graph, but as we will argue now, it is not a 1-bend EPG graph.

Consider a triangle in a 1-bend EPG representation. The union of three 1-bend paths cannot form a cycle in a rectangular grid, hence for each triangle the union of the paths forms a tree. It follows from the analysis of edge-intersection graphs of paths in trees [7] that these three paths either all intersect in one grid edge (we say that they are an *edge-clique*), or they form a *claw-clique*, i.e., there exists a $K_{1,3}$ in the grid such that any degree-1 vertex of the claw is used by exactly two paths. See Fig. 3(c).

Define a *3-sun* to be the graph that consists of a *central triangle*, and for every edge of the triangle a degree-2 vertex (called the *outrigger*) connected to the endpoints of the edge. From the work by Golumbic et al. [6], it follows that the central triangle of a 3-sun necessarily must be a claw-clique in any 1-bend EPG representation.

Now consider the triangle $\{a, b, c\}$ in graph G . Since it is part of an induced 3-sun, it is a claw-clique. After possible renaming, assume that a and b have a bend in common, and let a' be the outrigger-vertex attached to edge (a, b) . Since $\{a, b, a'\}$ is part of a sun, it, too, must be a claw clique. One of a and b must have a bend in that claw clique, too, say vertex a does. Since a has only one bend, the two bends in the two claw cliques must be at the same grid point, and hence a and b both have a bend in claw clique $\{a, b, a'\}$, and a' is drawn as a straight line. But then a' and c share grid edges: a contradiction since they are not adjacent. $\square$

As for upper bounds, it is well-known that every outer-planar graph has a vertex ordering $v_1, \dots, v_n$ such that each vertex v_i has at most 2 neighbours in $v_1, \dots, v_{i-1}$. We say that it is *2-regular acyclic orientable*. We can give a 3-bend construction for any graph that has such an orientation (which includes more graphs, for example all series-parallel graphs, and even some non-planar graphs such as the graph obtained from K_n by subdividing all edges.)

Theorem 6. *Every 2-regular acyclic orientable graph has 3-bend EPG representation in an $n \times n$ -grid.*

Proof. Let $v_1, \dots, v_n$ be a vertex order such that every vertex v_i has at most 2 earlier neighbours. We maintain the hypothesis that the graph induced by

$v_1, \dots, v_i$ has a 3-bend EPG representation such that every vertex-path contains a horizontal grid segment and a vertical grid segment that do not intersect any other vertex-path; we call these the *free* segments. This is trivial for v_1 : represent it by an arbitrary vertical and horizontal segment attached to each other.

To add vertex v_i to an existing drawing, find its earlier neighbours, say v_j and v_k . (The case of only one earlier neighbour is even easier.) Pick one grid edge e_j from the vertical free segment of v_j and one grid edge e_k from the horizontal free segment of v_k . Insert a new horizontal grid line ℓ_j that splits e_j , and a new vertical grid line ℓ_k that splits e_k .¹ Now define a path by using one part of e_j , going along ℓ_j until the crossing with ℓ_k , going along ℓ_k until e_k , and then using one part of e_k . See Fig. 4.

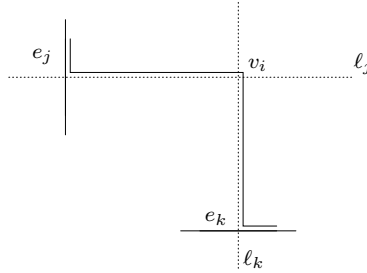


Fig. 4. The construction for 2-regular acyclic orientable graphs

One easily verifies that this path has three bends, intersects with the paths of v_j and v_k , and does not intersect any other paths. Moreover, v_j and v_k still have free segments (the other parts of e_j and e_k), and v_i also has free segments (along ℓ_j and ℓ_k), so the induction hypothesis holds with the new vertex. $\square$

We suspect that this construction is not optimal, especially for outer-planar graphs.

Conjecture 1. Every outer-planar graph is a 2-bend EPG graph.

5 Other Graph Classes

5.1 Line Graphs

A graph G is a line graph if there exists a root graph H such that every vertex of G corresponds to an edge of H , and two vertices of G are adjacent if the two corresponding edges have an endpoint in common. One can easily show the following result (we omit the proof but illustrate it in Figure 5):

Theorem 7. *Every line graph has a 2-bend EPG representation in an $n \times (m + 1)$ -grid.*

¹ “Inserting” a grid line really means that all coordinates of all endpoints and bends to the right/above are increased by 1.

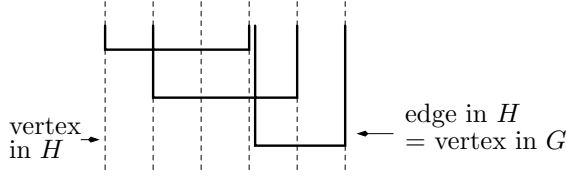


Fig. 5. 2-bend EPG representation of a line graph G with root graph H

5.2 Bounded Pathwidth Graphs

A graph of *pathwidth* k is a graph that has a vertex order $v_1, \dots, v_n$ such that for any j , at most k vertices in $v_1, \dots, v_{j-1}$ have a neighbour in $v_j, \dots, v_n$.

Theorem 8. *Every graph of pathwidth at most k has a $2k - 2$ -bend EPG representation in a $k \times (3n - k - 3)$ -grid.*

Proof. Let the vertex ordering $v_1, \dots, v_n$ be as described above. We say that vertex i is *active* at time $j > i$ if it has a neighbour in $\{v_j, \dots, v_n\}$; thus at most k vertices are active at any given time.

We route the vertices such that the paths of all vertices that are active at time j end in a ray to the right. v_1 simply starts as a horizontal line segment. For $j > 1$, consider the earlier neighbours of v_j . There can be at most k of them (since all of them are active at time j). Route v_j starting at the horizontal ray at one of its earlier neighbours, use one edge in common with that ray, use two bends to get to the ray of the next neighbour, and so on. The order of visiting the neighbours is irrelevant, except that the last one should be one that is not active at time $j + 1$, if there is such a neighbour.

Since at most k vertices are active at time $j + 1$, there are three possible cases (see also Fig. 6): (1) v_j has at most $k - 1$ earlier neighbours. Then we add another two bends to the route of v_j and let it end in a ray to the right. (2) v_j has k earlier neighbours, and v_j is not active at time $j + 1$. Then we end v_j 's path after the common grid edge with the last neighbour. (3) v_j has k earlier neighbours, and v_j is active at time $j + 1$. Then one of the earlier neighbours, say v_i , is not active at time $j + 1$, and v_j ends in its row. We let the grid path of v_i end at this point, and v_j continues as a rightward ray in the row that was v_i 's.

In all cases, we have used at most $2k - 2$ bends for v_j , and all paths of vertices active at time $j + 1$ end in a ray to the right. We can restrict the height by using only k rows, one for each of the active vertices. As for the width, the path of v_j uses only two columns except in case (3), where it uses four; but in case (3) the last column can be re-used for the path of v_{j+1} since it only contains one point from v_i and hence the paths of v_i and v_{j+1} are edge-disjoint. Since case (3) can only happen for $j \in \{k + 1, \dots, n - 1\}$ (v_j must have k earlier neighbours and at least one later neighbour), and since v_1 doesn't need to use a column (it can start when v_2 starts), the bound follows. $\square$

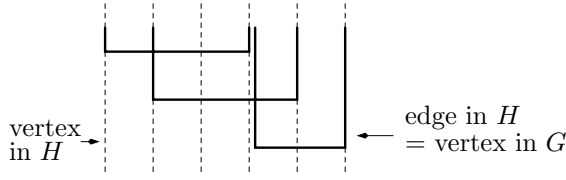


Fig. 6. The grid path for vertex v_j in the three cases of a graph of pathwidth k . Only grid paths of earlier neighbours of v_j are shown.

5.3 Graphs with Good Edge Orientations

We say that a graph has a κ -regular acyclic orientation if it has an acyclic edge orientation with $\text{indeg}(v) \leq \kappa$ for all vertices. The construction in [6] yields that every κ -regular acyclic orientable graph has a 2κ -bend EPG representation.

In Theorem 6, we improved this to 3 bends for $\kappa = 2$. We suspect that such an improvement is possible in general, and any κ -regular acyclic orientable graph is a $(2\kappa - 1)$ -bend EPG graph. Such graphs include graphs of treewidth κ (and also graphs of pathwidth κ , though Theorem 8 provides a better bound here.)

We say that a graph G has a κ -regular orientation if G has an edge orientation such that $\text{indeg}(v) \leq \kappa$ for all vertices v .

Theorem 9. *Every κ -regular orientable graph G has a $(2\kappa + 1)$ -bend EPG representation in a $2n \times 2n$ -grid. If G is bipartite, then the representation has only 2κ bends per path.*

Proof. (Sketch) Number the vertices arbitrarily as $v_1, \dots, v_n$. Initially each vertex is represented by an L (with 1 bend) using the $(2i)$ th column and the $(2n - 2i)$ th row (counting bottom-to-top) of an $2n \times 2n$ -grid. Any vertex v_i has a crossing with any vertex v_j ($j > i$) at point $(2i, 2n - 2j)$. If $v_i \rightarrow v_j$ is an edge, then re-route v_j near the crossing by adding two bends such that the path of v_i and v_j share an edge. This adds no bends to the path of v_i , so the bends for vertex v is $2\text{indeg}(v) + 1$, which is at most $2\kappa + 1$. See Fig. 7.

If G is bipartite, then let $\{v_1, \dots, v_a\}$ be one vertex class, and $\{v_{a+1}, \dots, v_n\}$ be the other. For each vertex then one part of the L can be omitted since it has no neighbours in this range; this saves one bend. $\square$

Graphs with a κ -regular orientation include planar graphs ($\kappa = 3$), and planar bipartite graphs ($\kappa = 2$), though we showed bounds better than Theorem 9 for these classes already. It is known that the smallest κ for which G has a κ -regular orientation is $\max_{H \subseteq G} \lceil |E(H)|/|V(H)| \rceil$ [2]. Also, every graph is $\lceil (\Delta + 1)/2 \rceil$ -regular orientable: edge-colour the graph with $\Delta + 1$ colours, split the graph into $\lceil (\Delta + 1)/2 \rceil$ subgraphs with maximum degree 2, and orient each, such that each vertex has at most one incoming edge. Edge colourings with Δ colours (which exist for bipartite graphs) gives even better bounds.

Corollary 1. *Every graph is a $2\lceil (\Delta + 1)/2 \rceil + 1$ -bend EPG graph. Every bipartite graph is a $2\lceil \Delta/2 \rceil$ -bend EPG graph.*

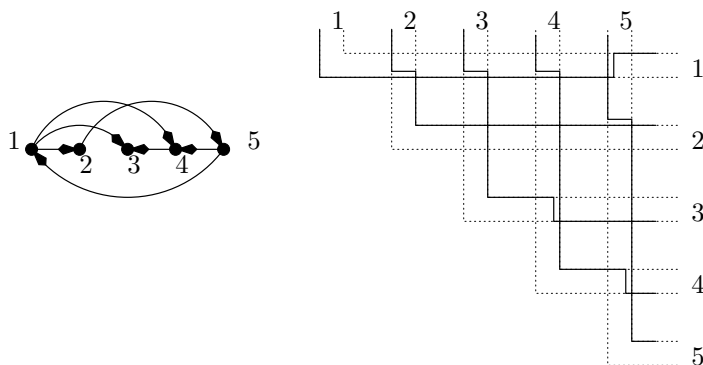


Fig. 7. 5-bend EPG representation of a 2-regular orientable graph

6 Remarks

We leave many open problems. An obvious one is to improve the upper or lower bounds for the number of bends for all graphs where this isn't tight yet. But more pressing are complexity issues. What is a recognition algorithm for 1-bend EPG graphs or k -bend EPG graphs? Is this NP-hard? What are time complexities of some problems in k -bend EPG graphs for small k ? Since planar graphs are 5-bend EPG graphs, the 3-Coloring problem is NP-hard in 5-bend EPG graphs. Is it polynomial for smaller k ?

References

1. Asinowski, A., Suk, A.: Edge intersection graphs of systems of grid paths with bounded number of bends. *Discrete Applied Mathematics* (accepted), preliminary version available at <http://www.technion.ac.il/~andrei/epg.pdf>
2. de Fraysseix, H., Ossona de Mendez, P.: Regular orientations, arboricity and augmentation. In: Tamassia, R., Tollis, I(Y.) G. (eds.) *GD 1994*. LNCS, vol. 894, pp. 111–118. Springer, Heidelberg (1995)
3. de Fraysseix, H., Ossona de Mendez, P., Pach, J.: Representation of planar graphs by segments. *Intuitive Geometry* 63, 109–117 (1991)
4. de Fraysseix, H., Ossona de Mendez, P., Rosenstiehl, P.: On triangle contact graphs. *Combinatorics, Probability and Computing* 3, 233–246 (1994)
5. Golumbic, M.C.: *Algorithmic graph theory and perfect graphs*, 2nd edn. Academic Press, New York (2004)
6. Golumbic, M.C., Lipshteyn, M., Stern, M.: Edge intersection graphs of single bend paths on a grid. In: *Sixth Cologne Twente Workshop on Graphs and Combinatorial Optimization (CTW 2007)*, University of Twente, pp. 53–55 (2007)
7. Golumbic, M.C., Lipshteyn, M., Stern, M.: The k -edge intersection graphs of paths in a tree. *Discrete Appl. Math.* 156(4), 451–461 (2008)
8. Nishizeki, T., Chiba, N.: *Planar Graphs: Theory and Algorithms*. North-Holland, Amsterdam (1988)

Efficient Data Structures for the Orthogonal Range Successor Problem*

Chih-Chiang Yu, Wing-Kai Hon, and Biing-Feng Wang

Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan 30043, Republic of China
{littlejohn,wkhon,bfwang}@cs.nthu.edu.tw

Abstract. This paper considers a type of orthogonal range query, called orthogonal range successor query, which is defined as follows: Let P be a set of n points that lie on an $n \times n$ grid. Then, for any given rectangle R , our target is to report, among all points of $P \cap R$, the point which has the smallest y -coordinate. We propose two indexing data structures for P so that online orthogonal range successor queries are supported efficiently. The first one is a *succinct* index where only $O(n)$ words are allowed for the index space. We show that each query can be answered in $O(\log n / \log \log n)$ time, thus improving the best-known $O(\log n)$ time by Mäkinen and Navarro. The improvement stems from the design of an index with $O(1)$ query time when the points are restricted to lie on a narrow grid, which in turn extends the recent wavelet tree technique to support the desired query. Our second result is a general framework for indexing points in the d -dimensional grids. We show an $O(n^{1+\epsilon})$ -space index that supports each d -dimensional query in optimal $O(1)$ time. Our second index is very simple and when $d = 2$, it is as efficient as the existing index by Crochemore *et al.*

Keywords: data structures, algorithms, range searching, indexes.

1 Introduction

Range searching problems have been intensively studied during the last 30 years, as they have many applications in a wide spectrum of areas such as database design, geographic information systems, computer graphics, and bio-informatics. In the most general setting, we are given a set of points P and a region Q in the d -dimensional space $\mathbb{R}^d$, and our target is to answer various queries about the set of points $P \cap Q$ (i.e., those points in P that are inside the region Q). Typical queries include the *range-reporting query*, in which all points in $P \cap Q$ are to be reported, and the *range-counting query*, in which only the number of points $|P \cap Q|$ is required. In addition, an *emptiness query* checks if $P \cap Q$ is empty and a *one-reporting query* reports an arbitrary point in $P \cap Q$ if there exists one. Sometimes, each point in P may be associated with a value (e.g., its distance

* This research was supported in part by the National Science Council of the Republic of China under the Contracts NSC-95-2213-E-007-029 and NSC-96-2221-E-007-082.

from the origin), and an *optimization query* can be defined to pick the point in $P \cap Q$ which is optimal under some criteria (e.g., to pick the point closest to the origin). When P and Q are both known, most queries can be answered simply by checking each point in P to see if it lies in Q . A more common situation is the *online* version of the problem where only P is given in advance, but we expect that range-searching queries on P will later be performed against many different regions Q . In this case, our target is to construct *indexing data structures* (or *indexes*) for P so that for any region Q given later, we can answer the queries efficiently. See the excellent survey by Agarwal and Erickson [1] for a reference.

In this paper, we focus on an optimization query called the *orthogonal range successor query*. This problem was introduced by Lenhof and Smid [9] and can be generally defined in the d -dimensional space $\mathbb{R}^d$ as follows: For a set of n d -dimensional points, on given any axis-parallel rectangle $R = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_{d-1}, b_{d-1}] \times [a_d, \infty]$, the target is to locate the point in R which has the smallest d th coordinate. As most of the existing work, we mainly focus on the 2-dimensional setting. For the 2-dimensional case, Lenhof and Smid [9] had an index with $O(n \log n)$ space and $O(\log n)$ query time.¹ The points in P in many applications will have integer coordinates; this motivates the study of the case where the points lie on an $n \times n$ grid. Keller *et al.* [8] extended the result of Lenhof and Smid by exploiting the grid property and improved the query time to $O(\log \log n)$. A more space-efficient index with optimal $O(n)$ space was obtained by Mäkinen and Navarro [10] whose query time is $O(\log n)$. When each point in P has distinct x - and y -coordinates, the orthogonal range successor problem on a 2-dimensional grid is exactly the *range next value problem* formulated by Crochemore *et al.* [4]. In the paper, Crochemore *et al.* gave an $O(n^2)$ -space index which supports each query in optimal $O(1)$ time. Later in [3], it was shown that by applying a multi-level scheme to the above index, an $O(n^{1+\varepsilon})$ -space index which supports each query in optimal $O(1)$ time can be obtained, for any constant $\varepsilon > 0$. The $O(n^2)$ -space index in [4] and the multi-level scheme in [3] both efficiently utilize the *range minimum query* [2]. Efficient indexes for the orthogonal range successor problem on a grid have numerous string matching applications, such as [4,7,10].

In this paper, we study optimal-space and optimal-query-time indexes for the orthogonal range successor problem on a grid. We first consider the *succinct* indexing problem, where only $O(n)$ words are allowed for the index space. We show that with a novel extension of the wavelet tree technique [5,6], each query can be answered in $O(\log n / \log \log n)$ time, thus improving the $O(\log n)$ time by Mäkinen and Navarro [10]. The proposed solution is essentially based on a restricted range successor index supporting $O(1)$ query time where all points in the index have bounded y -coordinates. It is worth mentioning that the $O(\log n / \log \log n)$ -time index also matches the best-known $O(n)$ -space one-reporting and emptiness index on a grid proposed by Nekrich [11]. Our second result is a general framework for indexing points in the d -dimensional grids, where $d \geq 2$ is a constant. For any fixed $\varepsilon > 0$, we show an $O(n^{1+\varepsilon})$ -space index

¹ All logarithms in this paper are base-2.

that supports each d -dimensional range successor query in optimal $O(1)$ time. Our second index is very simple and when $d = 2$, it is as efficient as the existing index by Crochemore *et al.* [3]. The results for orthogonal range successor queries are summarized in Table 1. Our computation model is a unit-cost RAM with word size $\log n$ bits. Unless stated otherwise, space is measured in words.

In Section 2, we introduce the notation and definitions. Then, we show our $O(n)$ -space orthogonal range successor index in Section 3, and give the general framework for d -dimensional index with optimal query time in Section 4.

Table 1. Orthogonal range successor query on a d -dimensional grid

	Construction Time	Space	Query time	Remarks
[10]	$O(n \log n)$	$O(n)$	$O(\log n)$	for $d = 2$
[8]	$O(n \log n \log \log n)$ expected	$O(n \log n)$	$O(\log \log n)$	
[3]	$O(n^{1+\varepsilon})$	$O(n^{1+\varepsilon})$	$O(1)$	
Ours	$O(n \log n / \log \log n)$	$O(n)$	$O(\log n / \log \log n)$	
Ours	$O(n^{1+\varepsilon})$	$O(n^{1+\varepsilon})$	$O(1)$	for any $d \geq 2$

2 Preliminaries

Let P be a set of n points in $[1, n] \times [1, n]$. For each point $p \in P$, its x -coordinate and y -coordinate are, respectively, denoted by $x(p)$ and $y(p)$. A *rectangle* is a cross product of two intervals. Given a point set P , the orthogonal range successor problem is to construct an index for P such that for any query rectangle $R \subseteq [1, n] \times [1, n]$, we can efficiently report the lowest point in P that lies in R . For ease of presentation, we abbreviate an orthogonal range successor index as an *RS index* and an orthogonal range successor query as an *RS query*.

Let S be an array whose length is denoted by $|S|$. The i th element of S is denoted by $S(i)$. For any indices i and j where $1 \leq i \leq j \leq |S|$, $S(i, j)$ consists of the elements $S(i), S(i+1), \dots, S(j)$.

2.1 Reduction to Range Successor on Integer Array

Let $(p(1), p(2), \dots, p(n))$ be the sequence obtained by sorting the points in P increasingly by the x -coordinates, breaking ties arbitrarily. Let A be an integer array of size n , where $A(i) = y(p(i))$ for $1 \leq i \leq n$. We can see that an RS query $[x_1, x_2] \times [y, \infty]$ on P is equivalent to finding the smallest element in $A(i_1, i_2)$ whose value is at least y , where i_1 is the smallest index with $x(p(i_1)) \geq x_1$ and i_2 is the largest index with $x(p(i_2)) \leq x_2$. In Sections 3 and 4, we give efficient indexes for A to support the latter query, thus solving the original problem.

2.2 Rank and Select Queries

Let S be a character string of length n over a finite, ordered alphabet $\Sigma = \{1, 2, \dots, |\Sigma|\}$. For any character $c \in \Sigma$ and any position i , a *rank query* $rank_c(S, i)$ reports the number of c in $S(1, i)$. A *select query* $select_c(S, j)$ returns the position of the j th occurrence of c in S . For instance, if $S = 231131321$, then $rank_3(S, 6) = 2$ and $select_3(S, 2) = 5$.

2.3 Wavelet Trees

Wavelet trees are elegant data structures introduced by Grossi *et al.* [6] for text indexing. Given a text S over alphabet Σ , the wavelet tree of S is a balanced binary tree of height $\log |\Sigma|$. Each tree node v corresponds to a subinterval $\Sigma_v \subseteq [1, |\Sigma|]$. The tree root corresponds to $[1, |\Sigma|]$. At each internal node, the current alphabet range is partitioned into two halves, and the corresponding alphabet subintervals are assigned to the left and right child of the node. A *subsequence* of S is a sequence obtained by deleting zero or more characters from S . Let S_v be the subsequence of S containing only the characters in the subinterval Σ_v . For example, if $S = 21831662$ and $\Sigma_v = [1, 4]$, then $S_v = 21312$. The only information stored at v is a bitmap B_v preprocessed for binary rank and select queries. For each character of S_v , it is indicated by B_v whether that character goes left or right. More specifically, $B_v(i) = 0$ if $S_v(i) \in \Sigma_{v(0)}$ and $B_v(i) = 1$ if $S_v(i) \in \Sigma_{v(1)}$, where $v(0)$ and $v(1)$ are, respectively, the left and right children of v . In Figure 1, we depict the wavelet tree of $S = 21831662$. Note that each S_v is listed for illustration only and it is not explicitly stored.

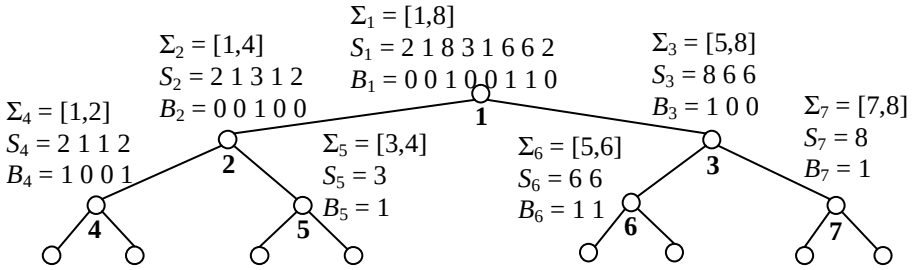


Fig. 1. The wavelet tree of $S = 21831662$

3 Succinct Index for Range Successor on Integer Array

This section describes a succinct index for the range successor problem on an integer array A . In Section 3.1, we first consider the special case where the values in A have a limited range of size $O(\log n / \log \log n)$, and give an index with optimal space and $O(1)$ query time. Then, in Section 3.2, we use the above index as a core and continue to present an index that handles the general case where values in the array range from 1 to n . The presented index uses optimal space and supports $O(\log n / \log \log n)$ query time.

3.1 The Core Index: Handling Ranges of Size $O(\log n / \log \log n)$

We define a range successor query on an integer array A as follows.

$\text{RS}(A, i, j, k)$: return the smallest value in $A(i, j)$ that is at least k ;
 return *null* if no such value exists.

The following establishes the core of our index. We shall show that given an array A of $m \leq n$ integers with range $r = O(\log n / \log \log n)$, we can construct an

$O(m \log r)$ -bit index so that together with an $o(n)$ -bit table, each RS query can be answered in $O(1)$ time. For ease of exposition, we assume $r \leq 0.25 \log n / \log \log n$. The result can be extended to $r = O(\log n / \log \log n)$. Also, we let $A(1, m)$ be the input array of integers, where each integer ranges from 1 to r .

We first describe a basic version of our RS index, whose space is $O(rm \log m)$ bits. The idea is to preprocess each query whose range is a power of two. For each position i ($1 \leq i \leq m$) and each choice of s ($0 \leq s \leq \log m$), we store a bitmap $B_{(i,s)}$ to indicate which integers are in $A(i, i + 2^s - 1)$. Precisely, $B_{(i,s)}$ is an r -bit vector whose w th bit is set to 1 if w is in $A(i, i + 2^s - 1)$, and is set to 0 otherwise. Similarly, we store a bitmap $B'_{(j,s)}$ to indicate which integers are in $A(j - 2^s + 1, j)$, where $1 \leq j \leq m$ and $0 \leq s \leq \log m$.

Consider answering an RS query on an arbitrary interval $[i, j]$. Let $s^* = \lfloor \log(j - i) \rfloor$. We can see that the (bitwise) union of $B_{(i,s^*)}$ and $B'_{(j,s^*)}$ gives a bitmap B_u indicating which integers are present in $A(i, j)$. Since each bitmap is of word size, the union operation can be done in $O(1)$ time. Consequently, $RS(A, i, j, k)$ can be answered by checking the position of the first 1 in $B_u(k, r)$. Since B_u is an r -bit vector and $k \in [1, r]$, we can construct an $O(2^r \times r \times \log r)$ -bit (i.e., $o(n)$ bits) table in advance so that the above checking can be done in $O(1)$ time by table look-up. In summary, each RS query can be answered in $O(1)$ time. For the space, the bitmaps require $O(rm \log m)$ bits in total as there are $O(m \log m)$ bitmaps, each taking r bits. Thus, we have the following result.

Lemma 1. *We can construct an $O(rm \log m)$ -bit index for $A(1, m)$ so that with an extra $o(n)$ -bit table, any RS query can be answered in $O(1)$ time. The construction times for the index and the table are $O(m \log m)$ and $o(n)$, respectively.*

Proof. The lemma follows since the bitmaps and the table can easily be constructed in the desired times by dynamic programming. $\square$

Next, we define a restricted form of an RS query as follows:

$RS_q(A, i, j, k)$: return $RS(A, i, j, k)$ if $(i \bmod q = 1)$ and $(j \bmod q = 0)$;
return *null* otherwise.

In other words, if we consider A to be partitioned into blocks of q integers, RS_q only supports RS queries with i and j being the block boundaries. Thus, to answer an RS_q query, it is sufficient to store $B_{(i,s)}$ and $B'_{(j,s)}$ only for those i and j being the block boundaries. Consequently, the index space and construction time can both be reduced by a factor of q . This gives the following corollary.

Corollary 1. *We can construct an $O((rm \log m)/q)$ -bit index for $A(1, m)$ so that with an extra $o(n)$ -bit table, any RS_q query can be answered in $O(1)$ time. The construction times for the index and the table are $O(m + (m \log m)/q)$ and $o(n)$, respectively.*

Now, we are ready to disclose the full version of our core index, which achieves space reduction by a standard multi-level scheme. In particular, we first partition the input array A into blocks of some specific size q_1 , say, $A_1, A_2, \dots, A_{m'}$, where

$m' = m/q_1$, and construct RS_q index for A with $q = q_1$ based on Corollary 1. Next, we further partition each block A_h into blocks of some specific size q_2 , and construct RS_q index for A_h with $q = q_2$. The partitioning process goes on until the size of each resulting block has size at most $b = 0.5 \log n / \log r$, in which case we can support $O(1)$ -time RS query in the block by maintaining a common table of $O(r^b \times b^2 r \log b)$ bits, which is $o(n)$ bits.

The values of $q_1, q_2, q_3, \dots$ are chosen carefully so that the total space of RS_q indexes at each level is bounded by $O(m \log r)$ bits. In general, at level t (for $t > 1$), we maintain m/q_{t-1} distinct RS_q indexes, each with $O((rq_{t-1} \log q_{t-1})/q_t)$ bits; this gives $O((rm \log q_{t-1})/q_t)$ bits in total. Thus, to achieve $O(m \log r)$ -bit space at each level, we set $q_1 = r \log m / \log r$, and for each subsequent q_t , where $t \geq 2$, we set $q_t = r \log q_{t-1} / \log r$. Consequently, since

$$\begin{aligned} q_2 &= r \log q_1 / \log r \\ &= r(\log r + \log \log m - \log \log r) / \log r \\ &< r(2 \log \log n) / \log r \quad (\text{since } r \leq 0.25 \log n \text{ and } m \leq n) \\ &\leq 0.5 \log n / \log r, \end{aligned}$$

only two levels of RS_q indexes are needed.

To answer $RS(A, i, j, k)$, we partition the interval $[i, j]$ into at most 5 subintervals, consisting of at most one subinterval of contiguous blocks at Level 1 (with block size q_1), two subintervals of contiguous blocks at Level 2 (with block size q_2), and two subintervals with at most q_2 integers at Level 3. (See Figure 2 for an illustration.) Then the desired value $RS(A, i, j, k)$ is simply the minimum of the range successors in these subintervals, where each can be computed in $O(1)$ time based on the RS_q indexes or table lookup.



Fig. 2. Partition $[i, j]$ to answer $RS(A, i, j, k)$

The construction time is analyzed as follows. By Corollary 1, the RS_q index at Level 1 is constructed in $O(m + (m \log m)/q_1)$ time. At Level 2, there are m/q_1 distinct RS_q indexes, each can be constructed in $O(q_1 + (q_1 \log q_1)/q_2)$ time. Thus, all the RS_q indexes at Level 2 can be constructed in $O(m + (m \log q_1)/q_2)$ time. Since $q_1 > \log m$ and $q_2 > \log q_1$, it is easy to conclude that the overall construction time is $O(m)$. With some efforts, we can adapt the above scheme to construct an index that supports $O(1)$ -time $rank_c$ query on A . Due to the page limitation, the details are omitted. We have the following.

Theorem 1. *We can construct an $O(m \log r)$ -bit index for $A(1, m)$ so that with an extra $o(n)$ -bit table, any RS or $rank_c$ query can be answered in $O(1)$ time. The construction times for the index and the table are $O(m)$ and $o(n)$, respectively.*

3.2 The General Index: Handling Ranges of Size n

Next, we consider the general case where values in the integer array $A(1, m)$ are chosen from a wider range $[1, n]$, and describe an $O(m)$ -space index that answers each $\text{RS}(A, i, j, k)$ query in $O(\log n / \log \log n)$ time. Our index is analogous to the wavelet tree in Section 2.3, except that we enlarge the branching factor from 2 to $\sqrt{\log n}$. Precisely, each node v in our wavelet tree T corresponds to a subinterval $\Sigma_v \subseteq [1, n]$ and represents the subsequence S_v of A whose integers are in Σ_v . For any internal node v , its $\sqrt{\log n}$ children are denoted by $v(1), v(2), \dots, v(\sqrt{\log n})$ and they partition Σ_v into $\sqrt{\log n}$ subintervals of equal size. The tree has height $2 \log n / \log \log n$ and the k th leaf represents the singleton interval $[k, k]$.

At each node v , we explicitly store a sequence B_v of length $|S_v|$ so that $B_v(i)$ indicates which child of v the integer $S_v(i)$ belongs to. Formally, $B_v(i) = j$ if $S_v(i) \in \Sigma_{v(j)}$. Furthermore, we store two auxiliary data structures described in Theorem 1 to support rank_c and RS queries. As the range of integers in B_v is $\sqrt{\log n} = O(\log n / \log \log n)$, both queries can be answered in $O(1)$ time, while the space is $O(|B_v| \log \sqrt{\log n}) = O(|B_v| \log \log n)$ bits. Thus, for indexing $A(1, m)$, each level of our wavelet tree takes $O(m \log \log n)$ bits. Since the tree height is $2 \log n / \log \log n$, the total space is $O(m \log n)$ bits, which is $O(m)$ words.

Based on the above data structures, we now describe how an $\text{RS}(A, i, j, k)$ query can be supported. The idea is first to determine whether k occurs in $A(i, j)$. If so, we can immediately report k as the desired answer. Otherwise, we will proceed to find and report the integer just larger than k in $A(i, j)$.

Let u be the root of T . To determine whether k occurs in $A(i, j)$, our strategy is to traverse the wavelet tree T from the root u to the leaf whose interval is $[k, k]$, and check along the way whether k actually occurs in $A(i, j)$. Precisely, let $u, v_1, v_2, \dots$ denote the nodes on the path; also, let k_t denote the rank of v_t among its siblings. That is, v_t is the k_t th child of its parent. Clearly, using the values of k and $\sqrt{\log n}$, all k_t can be computed in $O(\log n / \log \log n)$ time. We first examine in the root u if $B_u(i, j)$ contains k_1 , which can be done in $O(1)$ time by computing $\text{rank}_{k_1}(B_u, j) - \text{rank}_{k_1}(B_u, i - 1)$. If not, this implies no integer in $S_u(i, j)$ (i.e., $A(i, j)$) falls in the range of Σ_{v_1} , and we can conclude k does not occur in $A(i, j)$. Otherwise, we compute the contiguous portion $[i_1, j_1]$ of S_{v_1} that corresponds to the subsequence of $S_u(i, j)$ with values within the range of Σ_{v_1} , so that our problem is reduced to determine whether k is in $S_{v_1}(i_1, j_1)$. It is easy to check that $i_1 = \text{rank}_{k_1}(B_u, i - 1) + 1$ and $j_1 = \text{rank}_{k_1}(B_u, j)$, where these two values can be obtained in $O(1)$ time. Then, we proceed to visit v_1 and examine if $B_{v_1}(i_1, j_1)$ contains k_2 , which can be done in $O(1)$ time as before. If not, by a similar reasoning, we conclude k does not occur in $A(i, j)$. Otherwise, we compute a contiguous portion $[i_2, j_2]$ of S_{v_2} in $O(1)$ time and reduce our problem to determine whether k is in $S_{v_2}(i_2, j_2)$. In this way, determining whether k occurs in $A(i, j)$ can be done by traversing at most $2 \log n / \log \log n$ nodes in the wavelet tree, each taking $O(1)$ time based on our data structures.

Thus, it remains to show how to find $\text{RS}(A, i, j, k)$ in case k does not occur in $A(i, j)$. Let v_f be the first node such that there does not exist any integer k_{f+1} in $B_{v_f}(i_f, j_f)$ in the previous traversal. Our first step is to determine the

smallest value k' larger than k_{f+1} in $B_{v_f}(i_f, j_f)$, which can be computed in $O(1)$ time by an RS query on B_{v_f} . Then, there are two cases.

Case 1: Such a number k' exists. In this case, the desired answer must be an integer within the range of the k' th child of v_f . Let v' denote such a child node, and $[i', j']$ denote the contiguous portion of $S_{v'}$ that corresponds to the subsequence of $S_{v_f}(i_f, j_f)$ with values within the range of $\Sigma_{v'}$. Then, the desired answer is exactly the minimum value μ in $S_{v'}(i', j')$, which can be obtained by traversing the wavelet tree downwards from v' , and repeatedly refining the range that contains μ . The method is similar as before where we determine if k occurs in $A(i, j)$; more precisely, in each node visited, we issue two rank queries to update the contiguous portion of the sequence, but then use *one RS query* (instead of using a predetermined value k_t) to guide the next traversal. The total time is $O(\log n / \log \log n)$.

Case 2: No such k' exists. In this case, we go backwards along the path from v_f to u until reaching a node v_g such that $B_{v_g}(i_g, j_g)$ contains some integer k'' larger than k_{g+1} . If no such a node v_g exists, it is easy to see that all elements in $A(i, j)$ are smaller than k and the answer for our RS query is *null*. Otherwise, similar to Case 1, we conclude that the answer is the minimum value of some contiguous portion of sequence $S_{v''}$ (where $v'' = k''$ th child of v_g), which can be obtained in $O(\log n / \log \log n)$ time by the same strategy.

Consequently, we have the following.

Theorem 2. *We can construct an $O(m \log n)$ -bit index for an integer array $A(1, m)$ with values in $[1, n]$, where $m \leq n$, so that with an extra $o(n)$ -bit table, any RS query can be answered in $O(\log n / \log \log n)$ time. The construction times for the index and the table are $O(m \log n / \log \log n)$ and $o(n)$, respectively.*

4 A Simple Index with $O(1)$ Query Time

Let $\varepsilon > 0$ be an arbitrary constant. In Section 4.1, we give an $O(n^{1+\varepsilon})$ -space index which supports $O(1)$ query time for each RS query on $A(1, n)$. The presented index is an extension of the $O(n^{1+\varepsilon})$ -space range-reporting index obtained in [12], in which each query takes $O(\log n)$ time even though the input points have integral coordinates. In contrast, our scheme exploits the integral property and achieves optimal $O(1)$ time. Our index is simpler compared to the index by Crochemore *et al.* [3]. The result is further extended to higher dimensions $d \geq 3$ in Section 4.2. We first establish three simple results, whose proofs are omitted due to their simplicity.

Lemma 2. *Using $O(dm)$ space and $O(dm)$ preprocessing time, a range searching problem of m points in $[1, n]^d$ can be reduced in $O(d)$ time to a range searching problem in $[1, m]^d$, where $m < n$ is an integer.*

Lemma 3. *We can construct an index of size $O(n^3)$ in $O(n^3)$ time for $A(1, n)$ so that each RS query can be answered in $O(1)$ time.*

Corollary 2. *We can construct an index of size $O((n/q)^2 n)$ in $O((n/q)^2 n)$ time for $A(1, n)$ so that each RS_q query can be answered in $O(1)$ time.*

4.1 Optimal-Time 2-Dimensional Range Successor

To obtain our index, we partition the input array A into blocks of a specific size q_1 , say $A_1, A_2, \dots, A_\ell$ (with $\ell = n/q_1$), and construct RS_q index for A with $q = q_1$ based on Corollary 2. Each block A_h is further partitioned into blocks of some specific size q_2 and the RS_q index for A_h with $q = q_2$ is constructed. We continue the partitioning until the size of each block is small enough so that we can apply Lemma 3 to answer RS queries in those blocks at the last level.

Let $c = \lceil 2/\varepsilon \rceil$. Our intent is to choose the values of $q_1, q_2, \dots$ so that the space of RS_q indexes at each level is bounded by $O(n^{1+2/c})$. In addition, the RS indexes at the last level are also required to have space bounded by $O(n^{1+2/c})$. At the first level, we set $q_1 = n^{1-1/c}$. By Corollary 2, the RS_q index at level 1 uses $O(n^{1+2/c})$ space and supports $O(1)$ -time query. Consider the second level. There are n/q_1 distinct RS_q indexes at level 2, each with q_1 consecutive numbers of A ranging from 1 to n . To reduce the space of level 2, we perform range reduction over the q_1 numbers in each of the n/q_1 blocks at level 1, which by Lemma 2 requires a table of size $O(n^2/q_1) = O(n^{1+1/c})$. Then, the range of the numbers in each block at level 1 is reduced from $[1, n]$ to $[1, q_1]$. Thus, by Corollary 2, each RS_q index at level 2 uses $O((q_1/q_2)^2 q_1)$ space. By setting $q_2 = n^{1-2/c}$, the total space of RS_q indexes at level 2 is $O((n/q_1) \times (q_1/q_2)^2 q_1) = O(n^{1+2/c})$. Therefore, the data structures at level 2 occupy $O(n^{1+2/c})$ space in total.

In general, we set $q_t = n^{1-t/c}$ for $1 \leq t < c$. It is easy to see that the total space requirement of range reduction structures and RS_q indexes at each level t is bounded by $O(n^{1+1/c})$ and $O(n^{1+2/c})$, respectively. At level $c-1$, there are n/q_{c-1} blocks, each containing q_{c-1} numbers ranging from 1 to q_{c-1} . By Lemma 3, we construct RS indexes for the blocks at level $c-1$, using $O((n/q_{c-1}) \times q_{c-1}^3) = O(n \times q_{c-1}^2) = O(n \times n^{2-((2c-2)/c)}) = O(n^{1+2/c})$ space. Consequently, the total space of this multi-level RS index is $O(c \times n^{1+2/c}) = O(n^{1+\varepsilon})$.

The query process is similar to that in Section 3.1, whose details are deferred to the full paper. Then we have the following.

Theorem 3. *We can construct in $O(n^{1+\varepsilon})$ time an index of size $O(n^{1+\varepsilon})$ for $A(1, n)$ so that any RS query can be answered in $O(1)$ time.*

4.2 Extension to Higher Dimensions

It is easy to extend the scheme in Section 4.1 to any fixed $d \geq 3$ dimensions. Due to the structural similarities, only the 3-dimensional RS index is discussed. Let P be a set of n points in $[1, n]^3$. For each point $p \in P$, its x -, y -, and z -coordinates are, respectively, denoted by $x(p)$, $y(p)$, and $z(p)$. For ease of description, assume each point p has distinct x -coordinate. We redefine A to be an array in which $A(x(p))$ stores $(y(p), z(p))$ for each $p \in P$. An RS query $[x_1, x_2] \times [y_1, y_2] \times [z, \infty]$ is denoted by $\text{RS}(A, x_1, x_2, y_1, y_2, z)$.

To support RS queries, we construct for each pair (x_1, x_2) , $1 \leq x_1 \leq x_2 \leq n$, a 2-dimensional RS index for the points $\{(y(i), z(i)) \mid x_1 \leq i \leq x_2\}$. By Theorem 3, the above structures use $O(n^{3+\varepsilon})$ space and can answer each query RS query in $O(1)$ time. Then, we define RS_q query in three dimensions as follows:

$\text{RS}_q(A, x_1, x_2, y_1, y_2, z)$:
 return $\text{RS}(A, x_1, x_2, y_1, y_2, z)$ if $(x_1 \bmod q = 1)$ and $(x_2 \bmod q = 0)$;
 return *null* otherwise.

Using the structure in Theorem 3, we obtain an RS_q index of $O((n/q)^2 n^{1+\varepsilon})$ space which supports each RS_q query in $O(1)$ time. Then, we construct multi-level structures, using the same scheme in Section 4.1. Briefly speaking, our index has $4c - 1$ levels, where $c = \lceil 1/\varepsilon \rceil$, and each level spends $O(n^{1+1/4c})$ bits for the range reduction structures and $O(n^{1+1/c})$ bits for the RS_q indexes. The details are deferred to the full paper. Then we obtain the following.

Theorem 4. *Let P be a set of n points in $[1, n]^d$, where $d \geq 3$ is a constant. We can construct an RS index for P using $O(n^{1+\varepsilon})$ preprocessing time and $O(n^{1+\varepsilon})$ space so that each RS query can be answered in $O(1)$ time.*

References

1. Agarwal, P.K., Erickson, J.: Geometric Range Searching and Its Relatives. *Advances in Discrete and Computational Geometry* 223, 1–56 (1999)
2. Bender, M.A., Farach-Colton, M.: The LCA Problem Revisited. In: Gonnet, G.H., Viola, A. (eds.) *LATIN 2000*. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
3. Crochemore, M., Iliopoulos, C.S., Kubica, M., Rahman, M.S., Walen, T.: Improved Algorithms for the Range Next Value Problem and Applications. In: 25th Annual Symposium on Theoretical Aspects of Computer Science, pp. 205–216 (2008)
4. Crochemore, M., Iliopoulos, C.S., Rahman, M.S.: Finding Patterns in Given Intervals. In: Kučera, L., Kučera, A. (eds.) *MFCS 2007*. LNCS, vol. 4708, pp. 645–656. Springer, Heidelberg (2007)
5. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed Representations of Sequences and Full-Text Indexes. *ACM Transactions on Algorithms* 3(2) (2007)
6. Grossi, R., Gupta, A., Vitter, J.S.: High-Order Entropy-Compressed Text Indexes. In: 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 841–850 (2003)
7. Iliopoulos, C.S., Rahman, M.S.: Indexing Circular Patterns. In: Nakano, S.-i., Rahman, M.S. (eds.) *WALCOM 2008*. LNCS, vol. 4921, pp. 46–57. Springer, Heidelberg (2008)
8. Keller, O., Kopelowitz, T., Lewenstein, M.: Range Non-overlapping Indexing and Successive List Indexing. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) *WADS 2007*. LNCS, vol. 4619, pp. 625–636. Springer, Heidelberg (2007)
9. Lenhof, H.-P., Smid, M.H.M.: Using Persistent Data Structures for Adding Range Restrictions to Searching Problems. *RAIRO Theoretical Informatics and Application* 28(1), 25–49 (1994)
10. Mäkinen, V., Navarro, G.: Rank and Select Revisited and Extended. *Theor. Comput. Sci.* 387(3), 332–347 (2007)
11. Nekrich, Y.: Orthogonal Range Searching in Linear and Almost-Linear Space. *Comput. Geom.* 42(4), 342–351 (2009)
12. Preparata, F.P., Shamos, M.I.: *Computational Geometry: An Introduction*. Springer, Heidelberg (1985)

Reconstruction of Interval Graphs

Masashi Kiyomi, Toshiki Saitoh, and Ryuhei Uehara

School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan
{mkiyomi,toshikis,uehara}@jaist.ac.jp

Abstract. The graph reconstruction conjecture is a long-standing open problem in graph theory. There are many algorithmic studies related it besides mathematical studies, such as DECK CHECKING, LEGITIMATE DECK, PREIMAGE CONSTRUCTION, and PREIMAGE COUNTING. We study these algorithmic problems limiting the graph class to interval graphs. Since we can solve GRAPH ISOMORPHISM for interval graphs in polynomial time, DECK CHECKING for interval graphs is easily done in polynomial time. Since the number of interval graphs that can be obtained from an interval graph by adding a vertex and edges incident to it can be exponentially large, developing polynomial time algorithms for LEGITIMATE DECK, PREIMAGE CONSTRUCTION, and PREIMAGE COUNTING on interval graphs is not trivial. We present that these three problems are solvable in polynomial time on interval graphs.

Keywords: the graph reconstruction conjecture, interval graphs, polynomial time algorithm.

1 Introduction

Given a simple graph $G = (V, E)$, we call the multi-set $\{G - v \mid v \in V\}$ the deck of G where $G - v$ is a graph obtained from G by removing vertex v and incident edges. The graph reconstruction conjecture by Ulam and Kelly¹ is that for any multi-set D of graphs with at least two vertices there is at most one graph whose deck is D . We call a graph whose deck is D a preimage of D . No counter example is known for this conjecture, and there are many mathematical results about this conjecture. For example trees, regular graphs, and disconnected graphs are reconstructible (i.e. the conjecture is true for these classes) [5]. About interval graphs, Rimscha showed that interval graphs are recognizable in the sense that looking at the deck of G one can decide whether or not G belongs to interval graphs [10]. Rimscha also showed in the same paper that many subclasses of perfect graphs including perfect graphs themselves are recognizable, and some of subclasses including unit interval graphs are reconstructible. There are many good surveys about this conjecture. See for example [1,4].

Besides these mathematical results, there are some algorithmic results. We enumerate the algorithmic problems that we address in this paper.

¹ Determining the first person who proposed the graph reconstruction conjecture is difficult, actually. See [4] for the detail.

- Given a graph G and a multi-set of graphs D , check whether D is a deck of G (DECK CHECKING).
- Given a multi-set of graphs D , determine whether there is a graph whose deck is D (LEGITIMATE DECK).
- Given a multi-set of graphs D , construct a graph whose deck is D (PREIMAGE CONSTRUCTION).
- Given a multi-set of graphs D , compute the number of (pairwise nonisomorphic) graphs whose decks are D (PREIMAGE COUNTING).

Kratsch and Hemaspaandra showed that these problems are solvable in polynomial time for graphs of bounded degree, partial k -trees for any fixed k , and graphs of bounded genus, in particular for planar graphs [7]. In the same paper they proved many GI related complexity results.

There is a linear time algorithm for determining if given two interval graphs are isomorphic [9]. Thus developing a polynomial time algorithm for DECK CHECKING for interval graphs is easy.

Theorem 1. *There is an $O(n(n + m))$ time algorithm of DECK CHECKING for n -vertex m -edge graph and its deck (or a deck candidate) that consists of interval graphs.*

We will give the proof in Section 3.

LEGITIMATE DECK, PREIMAGE CONSTRUCTION, and PREIMAGE COUNTING for interval graphs are solvable by almost the same algorithm. In order to develop such an algorithm we show that given a set of n interval graphs D there are at most $O(n^2)$ graphs (preimages) whose decks are D . Further we can construct such $O(n^2)$ preimage candidates. Our algorithm checks these $O(n^2)$ candidates one by one whether its deck is D with DECK CHECKING algorithm. Our algorithm constructs n preimage candidates from $O(n)$ different interval representations of each interval graph in D by inserting an interval to them. The key is that the number of preimage candidates is $O(n^2)$ while a naive algorithm which inserts an interval to an interval representation may construct $\Omega(2^n)$ candidates (Consider the case that $\Theta(n)$ intervals terminate at some point t , and we insert a new left endpoint to t . The number of the ways of insertions is $\Theta(2^n)$ since there are $\Theta(n)$ choices whether the new interval intersects the old ones. Further, there may be many, say $\Theta(2^n)$, different compact interval representations for an interval graph. Therefore the number of preimage candidates will be very huge if we construct the candidates from all of them).

The following is our main theorem.

Theorem 2. *There are $O(n^3(n+m))$ time algorithms for LEGITIMATE DECK and PREIMAGE CONSTRUCTION, and there is an $O(n^4(n + m))$ time algorithm for PREIMAGE COUNTING, for n interval graphs.*

Note that m is the number of edges in the preimage. Kelly's lemma [5] shows that we can compute m from the deck.

We state terminologies in Section 2, then explain about interval graphs in Section 3. In Section 3 we introduce many small lemmas for those who unfamiliar

to interval graphs. Most of these lemmas may be well-known and/or basic for those who familiar to interval graphs and the notions of PQ-tree [2] and MPQ-tree [6]. However these lemmas play important roles in this paper. Then we show that the number of preimage candidates is $O(n^2)$, and we present our algorithm in Section 4. Finally we make some remarks in Section 5.

2 Terminology

Graphs in this paper are all simple and undirected, unless explicitly stated. We denote by $N_G[v]$ the closed neighbor set of vertex v in graph G . “Closed” means that $N_G[v]$ contains v itself. We denote by $\deg_G(v)$ the degree of vertex v in graph G . We omit the subscript G when there is no confusion about the base graph. The sum of degrees of all vertices in graph G is denoted by $\text{degsum}(G)$. Note that $\text{degsum}(G)$ is equal to twice the number of edges in G . We denote by $\tilde{G}$ the graph obtained by adding one universal vertex to the graph G such that the vertex connects to every vertex in G . Note that $\tilde{G}$ is always connected. Given two graphs G_1 and G_2 , we define the disjoint union $G_1 \dot{\cup} G_2$ of G_1 and G_2 as $(V_1 \dot{\cup} V_2, E_1 \dot{\cup} E_2)$ such that (V_1, E_1) is isomorphic to G_1 , and (V_2, E_2) is isomorphic to G_2 , where $\dot{\cup}$ means the disjoint union.

3 Interval Graphs

A graph $G = (V, E)$ with $V = \{v_1, v_2, \dots, v_n\}$ is an *interval graph* iff there is a multi-set $\mathcal{I} = \{I_{v_1}, I_{v_2}, \dots, I_{v_n}\}$ of closed intervals on the real line such that $\{v_i, v_j\} \in E$ if and only if $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $1 \leq i, j \leq n$. We call the multi-set $\mathcal{I}$ an *interval representation* of G . An interval graph may have infinitely many interval representations. We use a tractable one called compact interval representation among them.

3.1 Compact Representation and Basic Lemmas

Definition 1 ([11]). An interval representation $\mathcal{I}$ of an interval graph $G = (V, E)$ is compact iff

- coordinates of endpoints of intervals in $\mathcal{I}$ are finite non-negative integers (We denote by K the largest coordinates of endpoints for convenience. We sometimes call K the length of $\mathcal{I}$),
- there exists at least one endpoint whose coordinate is k for every integer $k \in [0, K]$, and
- interval multi-set $\mathcal{I}_k = \{I \in \mathcal{I} \mid k \in I\}$ differs from $\mathcal{I}_l = \{I \in \mathcal{I} \mid l \in I\}$, and they do not include each other, for every distinct integers $k, l \in [0, K]$.

We show an example of a compact interval representation of an interval graph in Fig. 1. Note that there may still be many compact interval representations of an interval graph. However compact interval representations have some good properties.

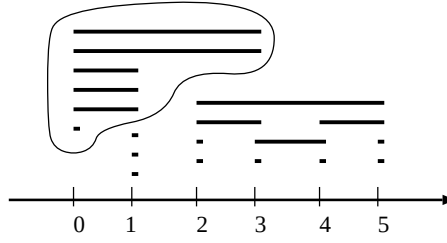


Fig. 1. A compact interval representation of an interval graph. Every interval graph has at least one compact interval representation. Vertices corresponding to the enclosed intervals are end-vertex set.

Lemma 1. Let $\mathcal{I}$ and $\mathcal{J}$ be compact interval representations of an interval graph $G = (V, E)$, and let K_1 be the length of $\mathcal{I}$, and let K_2 be the length of $\mathcal{J}$. Then the following holds.

$$\{\{I \in \mathcal{I} \mid 0 \in I\}, \{I \in \mathcal{I} \mid 1 \in I\}, \dots, \{I \in \mathcal{I} \mid K_1 \in I\}\} = \{\{I \in \mathcal{J} \mid 0 \in I\}, \{I \in \mathcal{J} \mid 1 \in I\}, \dots, \{I \in \mathcal{J} \mid K_2 \in I\}\}$$

Proof. We denote by $\bar{\mathcal{I}}$ the set of multi-set of intervals $\{\{I \in \mathcal{I} \mid 0 \in I\}, \{I \in \mathcal{I} \mid 1 \in I\}, \dots, \{I \in \mathcal{I} \mid K_1 \in I\}\}$, and we denote by $\bar{\mathcal{J}}$ the set of multi-set of intervals $\{\{I \in \mathcal{J} \mid 0 \in I\}, \{I \in \mathcal{J} \mid 1 \in I\}, \dots, \{I \in \mathcal{J} \mid K_2 \in I\}\}$. The vertices represented by the multi-set of intervals $\mathcal{I}_i = \{I \in \mathcal{I} \mid i \in I\}$ correspond to a clique in G . Assume that $\mathcal{I}_i$ never appears in $\bar{\mathcal{J}}$ for some i . Since $\mathcal{I}_i$ represents a clique C , there must be a set of intervals representing a clique C' containing C in $\bar{\mathcal{J}}$ (otherwise, clique C can not be represented in $\mathcal{J}$). Then for the same reason, $\bar{\mathcal{I}}$ must contain a set of intervals representing a clique containing C' . This contradicts the compactness of $\mathcal{I}$. $\square$

From the proof of Lemma 1, the following lemmas are straightforward.

Lemma 2. Let $\mathcal{I}$ be a compact interval representation of an interval graph $G = (V, E)$, and let K be the length of $\mathcal{I}$. Then $\{I \in \mathcal{I} \mid i \in I\}$ for each $i \in \{0, \dots, K\}$ corresponds to each maximal clique of G .

Lemma 3. The length of a compact interval representation of an n -vertex interval graph is at most n .

Note that the number of maximal cliques in an n -vertex interval graph is at most n (see [3]).

Lemma 4. All the compact interval representations of an interval graph have the same length.

Lemma 5. Intervals in different compact interval representations corresponding to an identical vertex have the same length.

From Lemma 5, lengths of intervals corresponding to a vertex that corresponds to an interval of length zero in some interval representation are always (i.e. in any interval representation) zero. Such a vertex is called *simplicial*.

```

boolean function deck-checking(graph  $G = (V, E)$ ) {
  Let  $G'$  be  $G$ .
  for each vertex  $v \in V$  {    $G' := G' \dot{\cup} (\tilde{G} - v)$ .   }
  if  $G'$  is isomorphic to  $\tilde{G}_1 \dot{\cup} \tilde{G}_2 \dot{\cup} \dots \dot{\cup} \tilde{G}_n \dot{\cup} G$ 
    return True   else   return False.
  }

```

Fig. 2. The deck checking algorithm

3.2 Deck Checking

Now we prove Theorem 1. Our main algorithm enumerates the preimage candidates, and checks whether each candidate is really a preimage of the input deck. Thus the theorem is one of the basic part of our algorithm. We first show a lemma to prove Theorem 1.

Lemma 6. *Given an interval graph G which can be connected or disconnected, $\tilde{G}$ is always a connected interval graph.*

Proof. It is obvious that $\tilde{G}$ is connected. Consider a compact interval representation $\mathcal{I}$ of G . Let K be the length of $\mathcal{I}$. Then $\mathcal{I} \cup \{[0, K]\}$ is an interval representation of $\tilde{G}$. Therefore $\tilde{G}$ is a connected interval graph. $\square$

Proof (Proof of Theorem 1). It is clear that $\{G_1, G_2, \dots, G_n\}$ is a deck of G if and only if $\{\tilde{G}_1, \tilde{G}_2, \dots, \tilde{G}_n\} \cup \{G\}$ is a deck of $\tilde{G}$. Hence we can determine whether or not the given multi-set $D = \{G_1, G_2, \dots, G_n\}$ is a deck of the input graph G by checking whether or not $\tilde{G}_1 \dot{\cup} \tilde{G}_2 \dot{\cup} \dots \dot{\cup} \tilde{G}_n \dot{\cup} G$ is isomorphic to the deck of $\tilde{G}$. Since the disjoint union of two interval graphs is an interval graph, we can use well-known linear time isomorphism algorithm [9] for this checking. We describe the algorithm in Fig. 2. Since the number of vertices of $\tilde{G}_1 \dot{\cup} \dots \dot{\cup} \tilde{G}_n \dot{\cup} G$ is $O(n^2)$, and since the number of edges of $\tilde{G}_1 \dot{\cup} \dots \dot{\cup} \tilde{G}_n \dot{\cup} G$ is $O(mn + n^2)$, the time complexity of this algorithm is $O(n(n + m))$. $\square$

3.3 Non-interval Graph Preimage Case

Our algorithm described in the next section outputs preimages that are interval graphs. However it is possible that a non-interval graph has a deck that consists of interval graphs, though it is exceptional. Since considering this case all the time in the main algorithm makes it complex, we attempt to get done with this special case.

First of this subsection we introduce a famous theorem below.

Theorem 3 (Lekkerkerker and Boland [8]). *Graph G is an interval graph if and only if G has no graph described in Fig. 3 as an induced subgraph.*

Note that we can easily prove that all the members in a deck of an interval graph are interval graphs from this lemma.

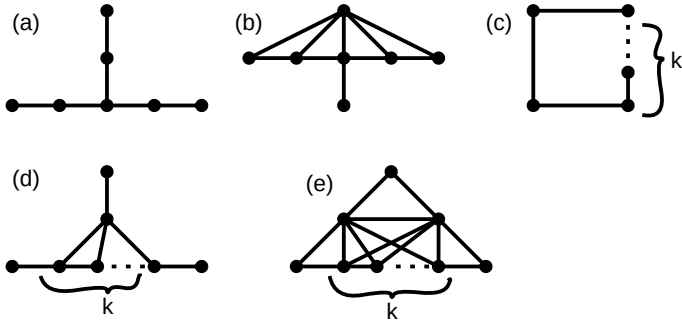


Fig. 3. The forbidden graphs of interval graphs. The part described k contains k vertices ($k \geq 1$). Thus (c) is a chordless cycle of more than three vertices, (d) has more than five vertices, and (e) has more than five vertices.

Theorem 4. *If n interval graphs $G_1, G_2, \dots, G_n$ have a preimage G that is not an interval graph, we can reconstruct G from $G_1, G_2, \dots, G_n$ in $O(n^2)$ time.*

Proof. Assume that $G_1, G_2, \dots, G_n$ are the deck of G , and G is not an interval graph. Then G must be one of the graphs described in Fig. 3, since any graph that is obtained by removing a vertex from G is an interval graph (containing none in Fig. 3). It is clear that $G_1, G_2, \dots, G_n$ have the same number of vertices, $n - 1$, and the number of vertices in G is n . Since the number of graphs of size n in Fig. 3 is $O(1)$, we can check if one of them is a preimage of the input graphs in polynomial time with DECK CHECKING algorithm. The time complexity is $O(n(n + m))$ from Theorem 1, where m is the number of edges of a preimage. Since the numbers of edges in (a), (b), (c), (d), and (e) are $O(n)$, the time complexity is definitely $O(n^2)$. $\square$

Therefore we concentrate on an algorithm that tries to reconstruct an interval graph whose deck is the set of the input graphs in the remaining sections.

4 Main Algorithm

4.1 Connected Preimage Case

It is possible that there is no connected preimage interval graph but a disconnected preimage interval graph of given n interval graphs. We consider this case later. Here in this subsection, we consider an algorithm for determining whether or not there are connected preimage interval graphs, and if any, returns them.

First we define end-vertex set. The end-vertex set is intuitively a set of vertices whose corresponding intervals are at the left end in an interval representation. Our algorithm adds a vertex adjacent to all the vertices in an end-vertex set of an interval graph in the input deck. This enables us to avoid exponential times' constructions of preimage candidates.

Definition 2. For an interval graph $G = (V, E)$, we call a vertex subset $S \subset V$ an *end-vertex set* iff in some compact interval representation of G all the coordinates of the left endpoints of intervals corresponding to vertices in S are 0, and S is maximal among such vertex subsets.

See Fig. 1 for example. It is clear from the definition of compact interval representations that an end-vertex set has at least one simplicial vertex.

We show some simple lemmas about end-vertex sets. We can estimate that the number of essentially different preimage candidates is $O(n^2)$ by these lemmas.

Lemma 7. Let S be an end-vertex set of an interval graph $G = (V, E)$. If two vertices v and w in S have the same degree, then $N[v]$ is equal to $N[w]$.

Proof. The statement is clear from the definition of compact interval representation (see Fig. 1 for the better understanding). $\square$

Lemma 8. A connected interval graph has at most $O(n)$ end-vertex sets.

Proof. An end-vertex set of an interval graph G is in the form $\{I \in \mathcal{I} \mid 0 \in I\}$ for some interval representation $\mathcal{I}$ of G . Thus, from Lemmas 1 and 3, there are at most $O(n)$ end-vertex sets for G . $\square$

Now we refer the well-known lemma about the degree sequence.

Lemma 9 (Kelly's Lemma [5]). We can calculate the degree sequence of a preimage of the input n graphs in $O(n)$ time, if we know the number of edges in each input graph.

Proof. Let $G_1, G_2, \dots, G_n$ be the input graphs. Assume that graph G has a deck $\{G_1, G_2, \dots, G_n\}$. Then there are vertices $v_1, v_2, \dots, v_n$ such that G_i is obtained by removing v_i from G for each i in $\{1, 2, \dots, n\}$. Thus

$$\text{degsum}(G_i) = \text{degsum}(G) - 2\text{deg}_G(v_i)$$

holds for each $i \in \{1, 2, \dots, n\}$. Hence we have

$$\text{degsum}(G) = \frac{\sum_{i=1}^n \text{degsum}(G_i)}{n - 2}.$$

Therefore we can easily calculate the degree sequence of G , i.e., $(\text{degsum}(G) - \text{degsum}(G_1))/2, (\text{degsum}(G) - \text{degsum}(G_2))/2, \dots, (\text{degsum}(G) - \text{degsum}(G_n))/2$.

We can calculate $\text{degsum}(G_i)$ in constant time, provided we know the number m_i of edges in G_i , for $\text{degsum}(G_i)$ is equal to $2m_i$. Thus the time complexity to calculate $\text{degsum}(G)$ is $O(n)$, and the total time complexity to obtain the degree sequence of G is also $O(n)$. $\square$

Now we present an algorithm for reconstructing a connected interval graph. Suppose that an n -vertex connected interval graph G has a deck of interval graphs $\{G_1, G_2, \dots, G_n\}$. Let $\mathcal{I}$ be a compact interval representation of G . There must be an index $i \in \{1, \dots, n\}$ such that G_i is obtained by removing a simplicial

```

for each  $G_i (i = 1, 2, \dots, n)$  {
  for each end-vertex set  $S'$  of  $G_i$  {

    Let  $\mathcal{I}$  be an interval representation of  $G_i$ 
    whose corresponding end-vertex set is  $S'$ .

    Compute the degree sequence  $(d_1, \dots, d_l)$  of  $S \setminus \{s\}$ .
    Let  $S''$  be a subset of  $S'$  whose degree sequence in  $G_i$  is  $(d_1 - 1, \dots, d_l - 1)$ .
    Let  $G$  be an interval graph
    whose interval representation is obtained from  $\mathcal{I}$ 
    by extending interval corresponding to vertices in  $S''$  to the left by one
    and adding an interval  $[-1, -1]$ .

    if deck-checking( $G$ ) = True
      output  $G$ .
  }
}
return No if the algorithm has output no graph.

```

Fig. 4. The algorithm for reconstructing connected interval graphs

vertex s in the end-vertex set S corresponding to $\mathcal{I}$. We want to reconstruct G from $G_1, \dots, G_n$. To do so, we first show that we can reconstruct G if we know the index i . Once we prove this, we can reconstruct G by checking if G_j is the desired G_i for every $j \in \{1, \dots, n\}$.

It is clear that $S \setminus \{s\}$ is contained in some end-vertex set S' of G_i . Of course we do not know S' if we do not know $S \setminus \{s\}$. However the number of the candidates of S' is $O(n)$ by Lemma 8. Thus checking if each candidate is S' can be done by $O(n)$ executions of the algorithm below. Let $\mathcal{I}_i$ be an interval representation of G_i whose corresponding end-vertex set is S' . Note that $\mathcal{I}_i$ is easily obtained in $O(n + m)$ time by using the data structure called MPQ-tree [6] if we know S' .

Now we try to specify $S \setminus \{s\}$. If we know $S \setminus \{s\}$, we can obtain G , since G has the interval representation obtained from $\mathcal{I}_i$ by extending intervals corresponding to vertices in $S \setminus \{s\}$ to the left by one and adding an interval $[-1, -1]$. Therefore we need to know $S \setminus \{s\}$. Since we know the degree sequence of G_i , and we can know the degree sequence of G by Lemma 9, we can know the degree sequence of $S \setminus \{s\}$. We denote the degree sequence by $(d_1, d_2, \dots, d_l)$. Now we can obtain $S \setminus \{s\}$; $S \setminus \{s\}$ is the subset of S' such that whose degree sequence in G_i is $(d_1 - 1, d_2 - 1, \dots, d_l - 1)$. Note that there may be many subsets of S' whose degree sequences in G_i are $(d_1 - 1, d_2 - 1, \dots, d_l - 1)$. However Lemma 7 guarantees that any of such subsets can be $S \setminus \{s\}$, i.e. all the graphs reconstructed in the assumption that some subset of S' whose degree sequence is $(d_1 - 1, d_2 - 1, \dots, d_l - 1)$ are $S \setminus \{s\}$ are isomorphic to each other. Therefore we can reconstruct G . The whole algorithm is described in Fig. 4.

Now we consider the time complexity of this algorithm. Because of the space limitation we omit the detail of the basic algorithms about MPQ-tree. For each G_i , calculating an MPQ-tree of G_i in $O(n + m)$ time helps us to list each S'

and I in $O(n)$ time. Computing the degree sequence $(d_1, d_2, \dots, d_l)$ takes $O(n)$ time from Lemma 9. Since obtaining S'' needs sorting of the degree sequence, it requires $O(n \log n)$ time. It is clear that reconstructing an interval graph from its interval representation takes $O(n + m)$ time, if the endpoints of intervals are sorted. DECK CHECKING algorithm costs $O(n(n + m))$. Therefore the total time complexity of this algorithm is $O(n((m+n) + n(n+m+n \log n + n(n+m)))) = O(n^3(m + n))$. Note that we have to check every output preimage is not isomorphic to each other for PREIMAGE COUNTING. Since the number of output preimage may be $O(n^2)$, we need $O(n^4(n + m))$ time for this checking. If the graph reconstruction conjecture is true, the time complexity of this checking can be omitted.

Theorem 5. *There is a polynomial time algorithm that lists up connected interval graphs that are preimages of the input n interval graphs. The time complexity for outputting one connected interval graph is $O(n^3(n + m))$, and that for outputting all is $O(n^4(m + n))$.*

4.2 Disconnected Preimage Case

Consider the case that the input graphs $G_1, G_2, \dots, G_n$ have a disconnected preimage G . Then from the argument in the Theorem 4, G must be an interval graph. Further it is proven that the graph reconstruction conjecture is true in this case [5]. Lemma 6 and the fact that $\{G_1, G_2, \dots, G_n\}$ is a deck of G if and only if $\{\tilde{G}_1, \tilde{G}_2, \dots, \tilde{G}_n\} \cup \{G\}$ is a deck of $\tilde{G}$ simplify our algorithm in this case.

Since we can know the degree sequence of G by Lemma 9, we can know the degree sequence of $\tilde{G}$ by Lemma 9. Thus we can obtain $\tilde{G}$ by the algorithm described in the previous subsection. Note that we do not know G , thus in fact we cannot use the algorithm itself. However in the algorithm we can omit the case that G_i in the algorithm is G , since every interval graph has at least two end-vertex set and so does $\tilde{G}$. Further we can omit checking if G is in the deck of $\tilde{G}$. If the new algorithm (omitting checking if G is in the deck of $\tilde{G}$) returns some $\tilde{G}$, we can construct G from it. Then now we can check if G is a preimage of $G_1, \dots, G_n$. Therefore we have the theorem below.

Theorem 6. *There is a polynomial time algorithm that outputs a disconnected interval graph that is the preimage of the input n interval graphs, if there exists. The time complexity of the algorithm is $O(n^3(m + n))$.*

Therefore we have the main theorem (Theorem 2) from Theorems 4, 5, and 6.

5 Concluding Remarks

The algorithms we described does not help directly the proof of the graph reconstruction conjecture on interval graphs. The conjecture on interval graphs remains to be open.

The complexities of graph reconstruction problems are strongly related to that of graph isomorphism problems. To develop a polynomial time algorithm

for a graph reconstruction problem restricting inputs to be in GI-hard graph class seems very hard. Since graph isomorphisms of circular-arc graphs are polynomial time solvable, circular-arc graph reconstruction problem may be a good challenge.

References

1. Bondy, J.A.: A graph reconstructor's manual. In: Surveys in Combinatorics. London Mathematical Society Lecture Note Series, vol. 166, pp. 221–252 (1991)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. Journal of Computer and System Sciences 13, 335–379 (1976)
3. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pacific Journal of Mathematics 15, 835–855 (1965)
4. Harary, F.: A survey of the reconstruction conjecture. Graphs and Combinatorics. Lecture Notes in Mathematics, vol. 406, pp. 18–28 (1974)
5. Kelly, P.J.: A congruence theorem for trees. Pacific Journal of Mathematics 7, 961–968 (1957)
6. Korte, N., Möhring, R.H.: An incremental linear-time algorithm for recognizing interval graphs. SIAM Journal on Computing 18, 68–81 (1989)
7. Kratsch, D., Hemaspaandra, L.A.: On the complexity of graph reconstruction. Mathematical Systems Theory 27, 257–273 (1994)
8. Leckerkerker, C.G., Boland, J.C.: Representation of a finite graph by a set of intervals on the real line. Fundamenta Mathematicae 51, 45–64 (1962)
9. Lueker, G.S., Booth, K.S.: A linear time algorithm for deciding interval graph isomorphism. Journal of the ACM 26, 183–195 (1979)
10. von Rimscha, M.: Reconstructibility and perfect graphs. Discrete Mathematics 47, 283–291 (1983)
11. Uehara, R., Uno, Y.: On computing longest paths in small graph classes. International Journal of Foundations of Computer Science 18, 911–930 (2007)

A Fast Algorithm for Computing a Nearly Equitable Edge Coloring with Balanced Conditions

Akiyoshi Shioura¹ and Mutsunori Yagiura²

¹ Graduate School of Information Sciences, Tohoku University,
Sendai 980-8579, Japan

shioura@dais.is.tohoku.ac.jp

² Graduate School of Information Science, Nagoya University,
Nagoya 464-8603, Japan
yagiura@nagoya-u.jp

Abstract. We discuss the nearly equitable edge coloring problem on a multigraph and propose an efficient algorithm for solving the problem, which has a better time complexity than the previous algorithms. The coloring computed by our algorithm satisfies additional balanced conditions on the number of edges used in each color class, where conditions are imposed on the balance among all edges in the multigraph as well as the balance among parallel edges between each vertex pair. None of the previous algorithms are guaranteed to satisfy these balanced conditions simultaneously.

1 Introduction

1.1 Problem Definition and Main Results

We discuss the nearly equitable edge coloring problem on a multigraph. Let $G = (V, E)$ be a multigraph; a multigraph is an undirected graph which may have parallel edges and/or loops. Throughout this paper, we denote by n and m the numbers of vertices and edges in G , respectively. Let $\mathcal{C} = \{1, 2, \dots, k\}$ be a set of k colors. An edge coloring of a multigraph G is an assignment of k colors to edges in E , which is represented by a function $\pi : E \rightarrow \mathcal{C}$.

Let $\pi : E \rightarrow \mathcal{C}$ be an edge coloring. For each vertex $v \in V$ and a color $i \in \mathcal{C}$, we denote by $d_\pi(v, i)$ the number of edges in E incident to v with color i . We say that an edge coloring π of a multigraph G is *nearly equitable* if it satisfies the condition

$$(\text{NEC}) \quad |d_\pi(v, i) - d_\pi(v, j)| \leq 2 \quad (\forall v \in V, \forall i, j \in \mathcal{C}).$$

The main aim of this paper is to propose a new algorithm for computing a nearly equitable edge coloring of a given multigraph. The time complexity of the proposed algorithm is better than the previous algorithms.

Table 1. Comparison of algorithms for the nearly equitable edge coloring problem. The mark “ $\sqrt{}$ ” means that the output of the algorithm satisfies (B1) and/or (B2).

authors	time complexity	(B1)	(B2)
Hilton & de Werra (1982) [4]	$O(km^2)$		$\sqrt{}$
Nakano et al. (1995) [7]	$O(m^2/k + mn)$		
Xie et al. (2004) [10]	$O(m^2/k)$	$\sqrt{}$	
Xie et al. (2008) [11]	$O(mn \log(m/(nk) + 1))$	$\sqrt{}$	
Ours	$O(\min\{mn, m^2/k\})$	$\sqrt{}$	$\sqrt{}$

In addition to the condition (NEC), we consider the following two “balanced” conditions on the number of edges used in each color class:

$$\begin{aligned} \text{(B1)} \quad & ||E_\pi^i| - |E_\pi^j|| \leq 1 \quad (\forall i, j \in \mathcal{C}), \\ \text{(B2)} \quad & ||E_\pi^i(u, v)| - |E_\pi^j(u, v)|| \leq 1 \quad (\forall i, j \in \mathcal{C}, \forall u, v \in V), \end{aligned}$$

where π is an edge coloring and

$$\begin{aligned} E_\pi^i &= \{e \in E \mid \pi(e) = i\} \quad (i \in \mathcal{C}), \\ E_\pi^i(u, v) &= \{e \in E \mid \pi(e) = i, e \text{ connects } u \text{ and } v\} \quad (i \in \mathcal{C}, u, v \in V). \end{aligned}$$

The first condition (B1) imposes that the number of all edges in each color class is almost the same, while the second condition (B2) imposes that each color class uses almost the same number of parallel edges between each pair of vertices. Note that (B1) is equivalent to the condition $|E_\pi^i| \in \{\lfloor m/k \rfloor, \lceil m/k \rceil\}$ ($\forall i \in \mathcal{C}$), while (B2) is equivalent to $|E_\pi^i(u, v)| \in \{\lfloor m(u, v)/k \rfloor, \lceil m(u, v)/k \rceil\}$ ($\forall i \in \mathcal{C}, \forall u, v \in V$), where $m(u, v)$ ($u, v \in V$) denotes the number of parallel edges connecting u and v . We show that the nearly equitable edge coloring computed by our algorithm satisfies both of the balanced conditions. Our main result is summarized as follows:

Theorem 1. *Our algorithm computes a nearly equitable edge coloring of a multigraph satisfying the conditions (B1) and (B2) in $O(\min\{mn, m^2/k\})$ time.*

Table 1 shows a summary of the previous algorithms for the nearly equitable edge coloring problem. The time complexity of our algorithm is better than the previous best bound $O(mn \log(m/(nk) + 1))$ by Xie et al. [11].¹ Moreover, our algorithm is the first to compute a nearly equitable edge coloring satisfying both of the conditions (B1) and (B2). The algorithms in [10,11] output a nearly equitable edge coloring satisfying (B1), and the output of the algorithm in [4] satisfies (B2), but none of the previous algorithms is guaranteed to obtain a coloring satisfying both of (B1) and (B2) (see Table 1).

¹ It is pointed out in Xie et al. [11] that $mn \log(m/(nk) + 1) = \Theta(m^2/k)$ holds for any m, n and k satisfying $0 < m/nk \leq 1$. From this fact it is not difficult to show that the algorithm in [11] is never asymptotically slower than that of [10], and that our new algorithm is never asymptotically slower than that of [11].

To compute a nearly equitable edge coloring, our algorithm iteratively modifies an edge coloring. For this, we propose a new recoloring procedure, which is based on a set of edge-disjoint alternating walks, while the previous algorithms are based on an Eulerian circuit [10,11] or a single alternating walk [4,7]. This recoloring procedure makes it possible to reduce the time complexity of the algorithm while keeping the conditions (B1) and (B2) of an edge coloring.

In the following discussion, we assume $k \leq m$ without loss of generality, since otherwise the problem is trivial.

1.2 Previous and Related Work

An edge coloring π of a multigraph G is said to be *equitable* if it satisfies the condition $|d_\pi(v, i) - d_\pi(v, j)| \leq 1$ ($\forall i, j \in \mathcal{C}, \forall v \in V$), which is stronger than the condition (NEC). Although every bipartite multigraph has an equitable edge coloring, non-bipartite multigraphs may not have an equitable edge coloring (see, e.g., [5,9]). A typical example is an odd cycle, which has no equitable edge coloring with $k = 2$. Several sufficient conditions for multigraphs to have an equitable edge coloring are shown in [4,5,8]. Note that the problem of determining the existence of an equitable edge coloring is NP-complete (see [11]).

The balanced conditions (B1) and (B2) have often been discussed in the literature of (nearly) equitable edge coloring [2,4,5,10,11]. The condition (B1) is referred to as “equalized condition” in [2] and “balanced condition” in [10,11], and (B2) is referred to as “edge-balanced condition” in [4].

Recently, a weighted version of the equitable edge coloring problem is discussed in [1,3], and the following conjecture for bipartite multigraphs is raised in [1]:

given a multigraph $G = (V, E)$, a set of colors $\mathcal{C} = \{1, 2, \dots, k\}$, and weights w_i ($i \in \mathcal{C}$) with $0 < w_i < 1$ and $\sum_{i \in \mathcal{C}} w_i = 1$, there exists an edge coloring such that

$$\lfloor w_i d(v) \rfloor \leq d_\pi(v, i) \leq \lceil w_i d(v) \rceil \quad (\forall i \in \mathcal{C}, \forall v \in V). \quad (1)$$

Note that the condition (1) coincides with the condition of equitable edge coloring if $w_i = 1/k$ for all $i \in \mathcal{C}$. The conjecture holds for some special cases, but does not hold in general, especially when G is not bipartite. The following relaxed statement where both of the upper and lower bounds are relaxed by two is proven for bipartite multigraphs in [1] and for general multigraphs in [3].

Theorem 2 ([1,3]). *Given a multigraph $G = (V, E)$, a set of colors $\mathcal{C} = \{1, 2, \dots, k\}$, and weights w_i ($i \in \mathcal{C}$) with $0 < w_i < 1$ and $\sum_{i \in \mathcal{C}} w_i = 1$, there exists an edge coloring such that $\lfloor w_i d(v) \rfloor - 2 \leq d_\pi(v, i) \leq \lceil w_i d(v) \rceil + 2$ for every $i \in \mathcal{C}$ and every $v \in V$.*

1.3 Overview of Our Algorithm

Our algorithm starts with an initial edge coloring satisfying (B1) and (B2), and repeatedly improves the edge coloring, without violating (B1) and (B2), so that it

satisfies the condition (NEC) in the end. As in many previous papers in the area of edge coloring, our algorithm improves an edge coloring by *switching edge colors of alternating walks* (see, e.g., [6]); the difference from the previous approach is that our algorithm uses a set of edge-disjoint alternating walks, not a single alternating walk, in each iteration. If a set of edge-disjoint alternating walks is chosen in a naive way, we can only show that the algorithm terminates in $O(m)$ iterations. To reduce the number of iterations, a set of edge-disjoint alternating walks is chosen in a deliberate way, which leads to the bound $O(\min\{kn, m\})$ on the number of iterations. We show that each iteration can be done in $O(m/k)$ time, and therefore the time complexity of the proposed algorithm is $O((m/k) \times \min\{kn, m\}) = O(\min\{mn, m^2/k\})$.

2 Switch of Edge Colors

The proposed algorithm modifies an edge coloring by using an operation called *switch*. For every distinct colors $\alpha, \beta \in \mathcal{C}$, we denote by $G_\pi(\alpha, \beta)$ the subgraph of G given by $G_\pi(\alpha, \beta) = (V, E_\pi^\alpha \cup E_\pi^\beta)$. Given an edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$, *switching edge colors of S* means to interchange the colors α and β of edges in S ; more formally, switching edge colors of S is to modify the current edge coloring $\pi : E \rightarrow \mathcal{C}$ to the new edge coloring $\pi' : E \rightarrow \mathcal{C}$ given by

$$\pi'(e) = \begin{cases} \beta & (e \in S, \pi(e) = \alpha), \\ \alpha & (e \in S, \pi(e) = \beta), \\ \pi(e) & (e \in E \setminus S). \end{cases}$$

To switch edge colors, the algorithm uses an edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$ satisfying the following condition:

$$\left. \begin{aligned} &\text{if } d_\pi(v, \alpha) \geq d_\pi(v, \beta), \text{ then } 0 \leq d_\pi^S(v, \alpha) - d_\pi^S(v, \beta) \leq d_\pi(v, \alpha) - d_\pi(v, \beta), \\ &\text{if } d_\pi(v, \alpha) \leq d_\pi(v, \beta), \text{ then } 0 \geq d_\pi^S(v, \alpha) - d_\pi^S(v, \beta) \geq d_\pi(v, \alpha) - d_\pi(v, \beta), \end{aligned} \right\} \quad (2)$$

where for each $v \in V$ and $i \in \{\alpha, \beta\}$, we denote by $d_\pi^S(v, i)$ the number of edges in S incident to v with color i . We say that S is *eligible* in the multigraph $G_\pi(\alpha, \beta)$ if it satisfies the condition (2) for all $v \in V$. Eligible edge sets are useful in getting a better edge coloring, as shown below.

Lemma 1. *Let $\pi : E \rightarrow \mathcal{C}$ be an edge coloring and $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$ an eligible edge set. Then, the new edge coloring $\pi' : E \rightarrow \mathcal{C}$ obtained by switching edge colors of S satisfies*

$$\begin{aligned} \min\{d_\pi(v, \alpha), d_\pi(v, \beta)\} &\leq \min\{d_{\pi'}(v, \alpha), d_{\pi'}(v, \beta)\} \\ &\leq \max\{d_{\pi'}(v, \alpha), d_{\pi'}(v, \beta)\} \leq \max\{d_\pi(v, \alpha), d_\pi(v, \beta)\} \quad (\forall v \in V). \end{aligned}$$

The proof is omitted due to space limitation. To keep balanced conditions (B1) and (B2), we consider the following two conditions for an edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$:

- (S1) if $|E_\pi^\alpha| = |E_\pi^\beta| + 1$, then $|S \cap E_\pi^\alpha| - |S \cap E_\pi^\beta| = 0$ or $+1$,
 if $|E_\pi^\alpha| = |E_\pi^\beta|$, then $|S \cap E_\pi^\alpha| - |S \cap E_\pi^\beta| = 0$,
 if $|E_\pi^\alpha| = |E_\pi^\beta| - 1$, then $|S \cap E_\pi^\alpha| - |S \cap E_\pi^\beta| = 0$ or -1 ,
 (S2) for every $u, v \in V$,
 if $|E_\pi^\alpha(u, v)| = |E_\pi^\beta(u, v)| + 1$, then $|S \cap E_\pi^\alpha(u, v)| - |S \cap E_\pi^\beta(u, v)| = 0$ or $+1$,
 if $|E_\pi^\alpha(u, v)| = |E_\pi^\beta(u, v)|$, then $|S \cap E_\pi^\alpha(u, v)| - |S \cap E_\pi^\beta(u, v)| = 0$,
 if $|E_\pi^\alpha(u, v)| = |E_\pi^\beta(u, v)| - 1$, then $|S \cap E_\pi^\alpha(u, v)| - |S \cap E_\pi^\beta(u, v)| = 0$ or -1 .

Lemma 2. *Let $\pi : E \rightarrow \mathcal{C}$ be an edge coloring, and $\pi' : E \rightarrow \mathcal{C}$ be the new edge coloring obtained by switching edge colors of an edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$.*

- (i) *If π and S satisfy (B1) and (S1), respectively, then π' satisfies (B1).*
 (ii) *If π and S satisfy (B2) and (S2), respectively, then π' satisfies (B2).*

The following is one of the key properties used in our algorithm. The proof will be given in Section 5.

Lemma 3. *Let $\pi : E \rightarrow \mathcal{C}$ be an edge coloring. Suppose that there exist two distinct colors $\alpha, \beta \in \mathcal{C}$ and a vertex $u \in V$ such that $d_\pi(u, \alpha) - d_\pi(u, \beta) \geq 3$ holds. For any integer $r \in \mathbb{Z}$ such that $1 \leq r \leq d_\pi(u, \alpha) - d_\pi(u, \beta) - 2$, we can compute an eligible edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$ satisfying the conditions (S1), (S2), and $d_\pi^S(u, \alpha) - d_\pi^S(u, \beta) \in \{r, r+1\}$ in $O(|E_\pi^\alpha \cup E_\pi^\beta|)$ time.*

3 Proposed Algorithm

We explain our algorithm for computing a nearly equitable edge coloring satisfying the conditions (B1) and (B2).

Our algorithm starts with an initial edge coloring satisfying (B1) and (B2), which can be easily computed in $O(m)$ time by using the following property.

Proposition 1. *Let $\{e_1, e_2, \dots, e_m\}$ be an ordered list of the edges in E such that the parallel edges connecting the same pair of vertices are ordered consecutively, and color each edge e_t ($t = 1, 2, \dots, m$) by the color $(t \bmod k) + 1$. Then, the resulting edge coloring satisfies the conditions (B1) and (B2).*

The algorithm always keeps the two conditions (B1) and (B2) satisfied, and iteratively improves the edge coloring so that the condition (NEC) is satisfied in the end.

To obtain an edge coloring π satisfying the condition (NEC), our algorithm processes each vertex $u \in V$ one by one. If the vertex u violates the condition

$$|d_\pi(u, i) - d_\pi(u, j)| \leq 2 \quad (\forall i, j \in \mathcal{C}), \quad (3)$$

then the algorithm repeatedly updates the edge coloring π by switching edge colors of an eligible edge set S until the condition (3) is satisfied. By Lemma 1, once the vertex u satisfies the condition (3), the edge coloring always satisfies (3) in the following iterations.

Suppose that the vertex u violates the condition (3). Our algorithm implicitly maintains the following sets of colors:

$$\mathcal{C}_\pi^0(u) = \{i \in \mathcal{C} \mid \lceil d(u)/k \rceil - 1 \leq d_\pi(u, i) \leq \lfloor d(u)/k \rfloor + 1\}, \quad (4)$$

$$\mathcal{C}_\pi^+(u) = \{i \in \mathcal{C} \mid d_\pi(u, i) \geq \lfloor d(u)/k \rfloor + 2\}, \quad (5)$$

$$\mathcal{C}_\pi^-(u) = \{i \in \mathcal{C} \mid d_\pi(u, i) \leq \lceil d(u)/k \rceil - 2\}. \quad (6)$$

Note that $\{\mathcal{C}_\pi^0(u), \mathcal{C}_\pi^+(u), \mathcal{C}_\pi^-(u)\}$ is a partition of $\mathcal{C}$. Whenever both of $\mathcal{C}_\pi^+(u)$ and $\mathcal{C}_\pi^-(u)$ are nonempty, the algorithm chooses two distinct colors α, β with $\alpha \in \mathcal{C}_\pi^+(u)$ and $\beta \in \mathcal{C}_\pi^-(u)$, which is done by choosing α and β satisfying $d_\pi(u, \alpha) = \max_{i \in \mathcal{C}} d_\pi(u, i)$ and $d_\pi(u, \beta) = \min_{i \in \mathcal{C}} d_\pi(u, i)$. Then, the algorithm updates the edge coloring π so that at least one of α and β is contained in $\mathcal{C}_\pi^0(u)$. This can be done efficiently by Lemma 3 with the value r given by

$$r = \min\{d_\pi(u, \alpha) - (\lfloor d(u)/k \rfloor + 1), (\lceil d(u)/k \rceil - 1) - d_\pi(u, \beta)\}. \quad (7)$$

Repeating these steps, we obtain either $\mathcal{C}_\pi^+(u) = \emptyset$ or $\mathcal{C}_\pi^-(u) = \emptyset$ (or both). Suppose that $\mathcal{C}_\pi^-(u) = \emptyset$ holds. Note that in this case, the right-hand side of (7) is nonpositive. Then, the algorithm iteratively updates the edge coloring π so that the value $\sum\{d_\pi(u, i) - \lceil d(u)/k \rceil \mid i \in \mathcal{C}_\pi^+(u)\}$ decreases at least by one while keeping the condition $\mathcal{C}_\pi^-(u) = \emptyset$. This is done by choosing two colors α and β with the same rule as above, and then using Lemma 3 with $r = 1$. In this way, the algorithm computes an edge coloring π satisfying (3).

Our algorithm is described as follows.

Algorithm. FASTBALANCING($G, \mathcal{C}$)

Input: a multigraph $G = (V, E)$ and a set of colors $\mathcal{C} = \{1, 2, \dots, k\}$.

Output: a nearly equitable edge coloring $\pi : E \rightarrow \mathcal{C}$ of G with (B1) and (B2).

1. Compute an initial edge coloring π satisfying the conditions (B1) and (B2).
2. **for** each $u \in V$ **do**
3. Compute the value $d_\pi(u, i)$ for all $i \in \mathcal{C}$.
4. **while** $\exists i, j \in \mathcal{C}$ such that $|d_\pi(u, i) - d_\pi(u, j)| \geq 3$ **do**
5. Compute colors $\alpha, \beta \in \mathcal{C}$ such that $d_\pi(u, \alpha) = \max_{i \in \mathcal{C}} d_\pi(u, i)$ and $d_\pi(u, \beta) = \min_{i \in \mathcal{C}} d_\pi(u, i)$.
6. Compute an eligible edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$ satisfying (S1), (S2), and $d_\pi^S(u, \alpha) - d_\pi^S(u, \beta) \in \{r, r + 1\}$, where r is given by $r = \max\{1, \min\{d_\pi(u, \alpha) - (\lfloor d(u)/k \rfloor + 1), (\lceil d(u)/k \rceil - 1) - d_\pi(u, \beta)\}\}$.
7. Modify the edge coloring π by switching edge colors of S .
8. **Output** π and **stop**.

We note that an eligible edge set S in Line 6 can always be obtained by Lemma 3. It is easy to see that the condition (NEC) is satisfied when the algorithm terminates. Since the edge set S chosen in Line 6 satisfies the conditions (S1) and (S2), the edge coloring π always satisfies (B1) and (B2) by Lemma 2. Hence, the output of the algorithm is a nearly equitable edge coloring satisfying (B1) and (B2).

4 Analysis of Time Complexity

We analyze the time complexity of the algorithm FASTBALANCING. First of all, we analyze the number of iterations of Lines 5–7 for a fixed vertex $u \in V$. For a real number z , we define a convex function $\varphi_z : \mathbb{R} \rightarrow \mathbb{R}$ by

$$\varphi_z(x) = \max\{\lfloor z \rfloor - x, 0, x - \lceil z \rceil\} \quad (x \in \mathbb{R}).$$

For an edge coloring $\pi : E \rightarrow \mathcal{C}$ and a vertex $u \in V$, we define

$$\Phi(\pi, u) = \sum_{i \in \mathcal{C}} \varphi_{d(u)/k}(d_\pi(u, i)).$$

The value $\Phi(\pi, u)$ is a nonnegative integer for every edge coloring π , and $\Phi(\pi, u) = 0$ holds if and only if $\lfloor d(u)/k \rfloor \leq d_\pi(u, i) \leq \lceil d(u)/k \rceil$ for all $i \in \mathcal{C}$. Thus, the value $\Phi(\pi, u)$ represents the degree of unbalance in the edge coloring π at the vertex u .

Lemma 4. *Let π be an edge coloring, $u \in V$ be a vertex, and $\alpha, \beta \in \mathcal{C}$ be distinct colors such that $d_\pi(u, \alpha) = \max_{i \in \mathcal{C}} d_\pi(u, i)$, $d_\pi(u, \beta) = \min_{i \in \mathcal{C}} d_\pi(u, i)$, and $d_\pi(u, \alpha) - d_\pi(u, \beta) \geq 3$. Suppose that π' is an edge coloring obtained by switching edge colors of an eligible edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$ with*

$$1 \leq d_\pi^S(u, \alpha) - d_\pi^S(u, \beta) \leq d_\pi(u, \alpha) - d_\pi(u, \beta) - 1. \quad (8)$$

Then, we have $\Phi(\pi', u) \leq \Phi(\pi, u) - 1$.

Proof. The proof is omitted due to space limitation. $\square$

Lemma 5. *For a fixed vertex $u \in V$, the number of iterations in the while loop in the algorithm FASTBALANCING is $O(d(u))$.*

Proof. The eligible set S computed in Line 6 satisfies the condition (8). Hence, the claim follows from Lemma 4 and the fact that $\Phi(\pi, u) = O(d(u))$. $\square$

Lemma 6. *For a fixed vertex $u \in V$, the number of iterations in the while loop in the algorithm FASTBALANCING is $O(k)$.*

Proof. In each iteration of the while loop, we consider the sets $\mathcal{C}_\pi^0(u), \mathcal{C}_\pi^+(u), \mathcal{C}_\pi^-(u)$ defined by (4), (5), and (6), respectively. Suppose that the colors α and β chosen in Line 5 satisfy $\alpha \in \mathcal{C}_\pi^+(u)$, $\beta \in \mathcal{C}_\pi^-(u)$. Recall that α and β are such that $d_\pi(u, \alpha) = \max_{i \in \mathcal{C}} d_\pi(u, i)$ and $d_\pi(u, \beta) = \min_{i \in \mathcal{C}} d_\pi(u, i)$. Let S be an edge set chosen in Line 6. Since the value r in Line 6 satisfies

$$r = \min\{d_\pi(u, \alpha) - (\lfloor d(u)/k \rfloor + 1), (\lceil d(u)/k \rceil - 1) - d_\pi(u, \beta)\} \geq 1,$$

at least one of α and β is contained in $\mathcal{C}_\pi^0(u)$ after switching edge colors of S . This fact implies that in at most k iterations, we have either $\mathcal{C}_\pi^+(u) = \emptyset$ or $\mathcal{C}_\pi^-(u) = \emptyset$.

Assume, without loss of generality, that $\mathcal{C}_\pi^-(u) = \emptyset$. Since $\sum\{d_\pi(u, i) - d(u)/k \mid i \in \mathcal{C}\} = d(u) - d(u) = 0$, it holds that

$$\begin{aligned} & \sum\{d_\pi(u, i) - d(u)/k \mid i \in \mathcal{C}, d_\pi(u, i) > d(u)/k\} \\ &= \sum\{d(u)/k - d_\pi(u, i) \mid i \in \mathcal{C}, d_\pi(u, i) \leq d(u)/k\}. \end{aligned}$$

Hence, we have

$$\begin{aligned} \Phi(\pi, u) &\leq \sum_{i \in \mathcal{C}} \max\{d_\pi(u, i) - d(u)/k, d(u)/k - d_\pi(u, i)\} \\ &= 2 \sum\{d(u)/k - d_\pi(u, i) \mid i \in \mathcal{C}, d_\pi(u, i) \leq d(u)/k\} \leq 2k, \end{aligned}$$

where the last inequality is by $\mathcal{C}_\pi^-(u) = \emptyset$. This fact, together with Lemma 4, implies that the while loop terminates in at most $2k$ iterations. $\square$

By Lemmas 5 and 6, the number of iterations of Lines 5–7 for a fixed vertex $u \in V$ is $O(\min\{k, d(u)\})$. We can compute an eligible edge set S satisfying the desired conditions in $O(|E_\pi^\alpha \cup E_\pi^\beta|) = O(m/k)$ time by Lemma 3. Switching edge colors in Line 7 requires $O(|S|) = O(m/k)$ time. Maintenance of values $d_\pi(u, i)$ and Line 5 can be done in $O(m)$ time in total by using a data structure shown in [11, Section 3]. Hence, the algorithm FASTBALANCING computes a nearly equitable edge coloring of a multigraph satisfying the conditions (B1) and (B2) in $O((m/k) \times \sum_{u \in V} \min\{k, d(u)\}) = O(\min\{mn, m^2/k\})$ time. This concludes the proof of Theorem 1.

5 Computing Eligible Edge Sets

In this section we give a proof of Lemma 3, which states that an eligible edge set $S \subseteq E_\pi^\alpha \cup E_\pi^\beta$ satisfying the conditions (S1), (S2), and an additional condition on the number $d_\pi^S(v, \alpha) - d_\pi^S(v, \beta)$ can be found in $O(|E_\pi^\alpha \cup E_\pi^\beta|)$ time. To prove this, we consider a decomposition of the edge set $E_\pi^\alpha \cup E_\pi^\beta$ by using eligible alternating walks to be defined below.

A *walk* is a sequence of vertices and edges of the form $u_0 e_1 u_1 e_2 u_2 \dots e_{t-1} u_{t-1} e_t u_t$, where $u_0, u_1, \dots, u_t$ are vertices and $e_1, e_2, \dots, e_t$ are distinct edges such that e_j connects the vertices u_{j-1} and u_j for $j = 1, 2, \dots, t$. It is mentioned that a walk may visit the same vertex more than once; in particular, it is possible that the first and last vertices u_0 and u_t are the same. A walk is said to be *eligible* if the set of all edges in the walk is eligible. In the following discussion, we may regard a walk as the set of edges $\{e_1, e_2, \dots, e_t\}$ to simplify the description.

Let $\pi : E \rightarrow \mathcal{C}$ be an edge coloring, and $\alpha, \beta \in \mathcal{C}$ distinct colors. We call a walk P in the multigraph $G_\pi(\alpha, \beta)$ an *alternating walk* if any two consecutive edges in P have different colors. Alternating walks in $G_\pi(\alpha, \beta)$ can be categorized into the following three types. An $\alpha\beta$ -*even alternating walk* is an alternating walk P such that $|P \cap E_\pi^\alpha| = |P \cap E_\pi^\beta|$. An α -*odd alternating walk* (resp., a β -*odd alternating walk*) is an alternating walk P such that $|P \cap E_\pi^\alpha| = |P \cap E_\pi^\beta| + 1$

(resp., $|P \cap E_\pi^\beta| = |P \cap E_\pi^\alpha| + 1$). In the following, we mainly consider eligible alternating walks in $G_\pi(\alpha, \beta)$.

Lemma 7 ([6,7]). *Let $u_0 \in V$ be a vertex such that $d_\pi(u_0, \alpha) \neq d_\pi(u_0, \beta)$. Then, there exists an eligible alternating walk $P = u_0 e_1 u_1 e_2 u_2 \dots e_{t-1} u_{t-1} e_t u_t$ starting from u_0 .*

A partition $\{P_1, P_2, \dots, P_s, R\}$ ($s \geq 0$) of the edge set $E_\pi^\alpha \cup E_\pi^\beta$ of the multigraph $G_\pi(\alpha, \beta)$ is called an *alternating walk decomposition* if P_h ($h = 1, 2, \dots, s$) are eligible alternating walks satisfying the following condition:

$$\sum_{h=1}^s \{d_\pi^{P_h}(v, \alpha) - d_\pi^{P_h}(v, \beta)\} = d_\pi(v, \alpha) - d_\pi(v, \beta) \quad (\forall v \in V). \quad (9)$$

Note that an alternating walk decomposition is not uniquely determined. An alternating walk decomposition always exists, and can be obtained by the following algorithm.

Step 0: Set $s := 0$ and $E' := E_\pi^\alpha \cup E_\pi^\beta$.

Step 1: If $d_\pi^{E'}(v, \alpha) = d_\pi^{E'}(v, \beta)$ ($\forall v \in V$), then output $\{P_1, P_2, \dots, P_s, E'\}$ and stop.

Step 2: Let $v \in V$ be a vertex with $d_\pi^{E'}(v, \alpha) \neq d_\pi^{E'}(v, \beta)$.

Step 3: Find an eligible alternating walk P_{s+1} in the multigraph (V, E') starting from v .

Step 4: Set $E' := E' \setminus P_{s+1}$ and $s := s + 1$. Go to Step 1.

It is not difficult to implement this algorithm so that it runs in $O(|E_\pi^\alpha \cup E_\pi^\beta|)$ time.

We now prove Lemma 3. Suppose that there exist two distinct colors $\alpha, \beta \in \mathcal{C}$ and a vertex $u \in V$ such that $d_\pi(u, \alpha) - d_\pi(u, \beta) \geq 3$. Let $\{P_1, P_2, \dots, P_s, R\}$ be an alternating walk decomposition of $E_\pi^\alpha \cup E_\pi^\beta$. In the following, we show that there exists a subset $\mathcal{P} \subseteq \{P_1, P_2, \dots, P_s\}$ of alternating walks such that the set $S = \bigcup_{P \in \mathcal{P}} P$ satisfies the conditions (S1), (S2), and

$$d_\pi^S(u, \alpha) - d_\pi^S(u, \beta) \in \{r, r + 1\}, \quad (10)$$

where r is an integer with $1 \leq r \leq d_\pi(u, \alpha) - d_\pi(u, \beta) - 2$. We note that for any $\mathcal{P} \subseteq \{P_1, P_2, \dots, P_s\}$, the set $S = \bigcup_{P \in \mathcal{P}} P$ is eligible since $\{P_1, P_2, \dots, P_s, R\}$ is an alternating walk decomposition. The proof given below is constructive, and it immediately yields an algorithm for computing an eligible edge set satisfying the desired conditions in $O(|E_\pi^\alpha \cup E_\pi^\beta|)$ time.

We first consider the condition (10). We assume that $P_1, \dots, P_{s'}$ ($s' \geq 0$) are the alternating walks such that both of the end vertices are u , and $P_{s'+1}, \dots, P_{s''}$ ($s'' \geq s'$) are the alternating walks such that only one of the end vertices is u . We start with $\mathcal{P} = \emptyset$, and add the walks $P_1, P_2, \dots, P_{\min\{s', \lceil r/2 \rceil\}}$ to the set $\mathcal{P}$. If $s' \geq \lceil r/2 \rceil$, then the edge set $S = \bigcup_{P \in \mathcal{P}} P$ satisfies $d_\pi^S(u, \alpha) - d_\pi^S(u, \beta) = 2\lceil r/2 \rceil \in \{r, r + 1\}$; i.e., (10) holds. Otherwise (i.e., $s' < \lceil r/2 \rceil$), we further add

the walks $P_{s'+1}, P_{s'+2}, \dots, P_{s'+(r-2s')}$ to $\mathcal{P}$. Then, $S = \bigcup_{P \in \mathcal{P}} P$ satisfies (10). We note that $s' + (r - 2s') \leq s''$ holds since $2s' + (s'' - s') = d_\pi(u, \alpha) - d_\pi(u, \beta) > r$.

We then consider the property (S1). We note that none of walks in the current set $\mathcal{P}$ is a β -odd alternating walk since every eligible alternating walk starting from the vertex u is either an $\alpha\beta$ -even alternating walk or an α -odd alternating walk. Let t_α be the number of α -odd alternating walks in $\mathcal{P}$, and define t_β by $t_\beta = \max\{0, t_\alpha - 1\}$ if $|E_\pi^\alpha| = |E_\pi^\beta| + 1$, and $t_\beta = t_\alpha$ otherwise. We see from the following simple observation that the number of β -odd alternating walks in $\{P_1, P_2, \dots, P_s\}$ is at least t_β .

Lemma 8. *Let $\{P_1, P_2, \dots, P_s, R\}$ be an alternating walk decomposition of $E_\pi^\alpha \cup E_\pi^\beta$, and let s_α (resp., s_β) be the number of α -odd (resp., β -odd) alternating walks in $\{P_1, P_2, \dots, P_s\}$. Then, we have $s_\alpha - s_\beta = |E_\pi^\alpha| - |E_\pi^\beta|$.*

We choose t_β β -odd alternating walks in the decomposition arbitrarily and add them to $\mathcal{P}$. Note that u cannot be an end vertex of a β -odd alternating walk, and hence the addition of β -odd alternating walks does not affect the condition (10). Therefore, the edge set $S = \bigcup_{P \in \mathcal{P}} P$ satisfies both of (10) and (S1).

Finally, we consider the condition (S2). We use a similar technique as in [4,6]. Let $G_\pi^*(\alpha, \beta)$ be a subgraph of $G_\pi(\alpha, \beta)$ defined as follows. From the multigraph $G_\pi(\alpha, \beta)$, delete successively all pairs of edges of color α and β respectively connecting the same two vertices as far as such a pair of edges exists, and let $G_\pi^*(\alpha, \beta) = (V, E^*)$ be the resulting multigraph. Obviously, for each pair of vertices v, v' there exists at most one edge connecting v and v' ; an edge (v, v') with color α (resp., β) is in E^* if and only if $|E_\pi^\alpha(v, v')| = |E_\pi^\beta(v, v')| + 1$ (resp., $|E_\pi^\beta(v, v')| = |E_\pi^\alpha(v, v')| + 1$). Hence, any subset S of E^* satisfies the condition (S2). This means that if we consider an edge set of the graph $G_\pi^*(\alpha, \beta)$ instead of the original graph $G_\pi(\alpha, \beta)$, the condition (S2) is automatically satisfied. This modification does not affect (S1) since $G_\pi^*(\alpha, \beta)$ is obtained by removing the same number of edges from E_π^α and from E_π^β . Moreover, we have $d_\pi^{E^*}(v, \alpha) - d_\pi^{E^*}(v, \beta) = d_\pi(v, \alpha) - d_\pi(v, \beta)$ ($\forall v \in V$). This implies that the conditions concerning the balance around each vertex such as eligibility condition (2) and the conditions (9) and (10) are not affected by the replacement of $G_\pi(\alpha, \beta)$ with $G_\pi^*(\alpha, \beta)$. In summary, this replacement of the multigraph does not affect the properties shown in the previous discussion. This concludes the proof of Lemma 3.

References

1. Correa, J., Goemans, M.X.: Improved bounds on nonblocking 3-stage Clos networks. SIAM J. Comput. 37, 870–894 (2007)
2. Dugdale, J.K., Hilton, A.J.W.: Amalgamated factorizations of complete graphs. Combin. Probab. Comput. 3, 215–231 (1994)
3. Feige, U., Singh, M.: Edge coloring and decompositions of weighted graphs. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 405–416. Springer, Heidelberg (2008)

4. Hilton, A.J.W., de Werra, D.: Sufficient conditions for balanced and for equitable edge-colouring of graphs. O.R. Working paper 82/3, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Switzerland (1982)
5. Hilton, A.J.W., de Werra, D.: A sufficient condition for equitable edge-colourings of simple graphs. *Discrete Math.* 128, 179–201 (1994)
6. Nakano, S., Nishizeki, T.: Scheduling file transfers under port and channel constraints. *Internat. J. Found. Comput. Sci.* 4, 101–115 (1993)
7. Nakano, S., Suzuki, Y., Nishizeki, T.: An algorithm for the nearly equitable edge-coloring of graphs. *IEICE Trans. Inf. & Syst.* J78-D-I, 437–444 (1995) (in Japanese)
8. Song, H., Wu, J., Liu, G.: The equitable edge-coloring of series-parallel graphs. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) *ICCS 2007*. LNCS, vol. 4489, pp. 457–460. Springer, Heidelberg (2007)
9. de Werra, D.: Equitable colorations of graphs. *Revue française d’Informatique et de Recherche Operationelle R-3*, pp. 3–8 (1971)
10. Xie, X., Ono, T., Nakano, S., Hirata, T.: An improved algorithm for the nearly equitable edge-coloring problem. *IEICE Trans. Fund.* E87-A, 1029–1033 (2004)
11. Xie, X., Yagiura, M., Ono, T., Hirata, T., Zwick, U.: An efficient algorithm for the nearly equitable edge coloring problem. *J. Graph Algorithms Appl.* 12, 383–399 (2008)

Minimal Assumptions and Round Complexity for Concurrent Zero-Knowledge in the Bare Public-Key Model

Giovanni Di Crescenzo

Telcordia Technologies, Piscataway, NJ, USA
giovanni@research.telcordia.com

Abstract. Under the (*minimal*) assumption of the existence of one-way functions, we show that every language in NP has (*round-optimal*) argument systems in the bare public key (BPK) model of [3], which are sound (i.e., a cheating prover cannot prove that $x \notin L$) and (black-box) zero-knowledge (i.e., a cheating verifier does not obtain any additional information other than $x \in L$) even in the presence of concurrent attacks (i.e., even if the cheating prover or verifier are allowed to arbitrarily interleave several executions of the same protocol). This improves over the previous best result [12], which obtained such a protocol using a stronger assumption (the existence of one-way permutations) or a higher round complexity (5 messages), and is round-optimal among black-box zero-knowledge protocols. We also discuss various extensions and applications of our techniques with respect to protocols with different security and efficiency requirements.

Keywords: Zero-Knowledge Protocols, Concurrent Zero-Knowledge, Bare Public-Key Model, Complexity Assumptions, Round Complexity.

1 Introduction

The classical notion of a zero knowledge proof (a proof that reveals no additional information other than the theorem being true, even to malicious verifiers) was introduced in [18] and has been of interest, since its introduction, both in computational complexity and in the design of cryptographic protocols. Motivated by applications in networks like the Internet several researchers realized the need of extending the security properties of zero-knowledge protocols. Concurrent zero-knowledge, as formally defined in [15], considers the case of several concurrent executions of the same protocol, where a malicious adversary may control the scheduling of the messages and corrupt multiple provers or verifiers in order to violate the soundness or zero-knowledge properties. This notion has been studied in a (standard) interactive protocol model of [18] without additional setup infrastructures or network assumptions, where several protocols have been proposed [21,25], but super-constant lower bounds on the round complexity have been given [4]. As these bounds severely limit the applicability of these protocols and this notion, other models are being studied to achieve efficient, and, in particular, constant-round concurrent zero-knowledge protocols. In all these models, the most desired result is establishing, under general complexity assumptions (ideally, the existence of one-way functions), the existence of a zero-knowledge protocol for proving membership to any

language in NP (as this is typically required by most cryptographic applications of zero-knowledge), which additionally has efficient implementations (in terms of time, round and communication complexity).

In this paper we consider the bare public-key (BPK) model from [3], as it seems to require minimal setup or network assumptions to obtain such results. In this model, it is postulated that verifiers register their public key in a public file during a setup stage, and there is no interactive preprocessing stage, trusted string or third party, or assumption on the asynchronicity of the network. In this model, the concurrent soundness and zero-knowledge notions are harder to achieve than their weaker variants, as noted in [22], who discussed four distinct and increasingly stronger soundness notions: one-time, sequential, concurrent and resettable soundness. Several constant-round concurrent and resettable zero-knowledge protocols have been presented in the BPK model; all these protocols either make additional model assumptions on top of the BPK model [23,11,29], or do not satisfy concurrent soundness [3,9], or require complexity assumptions against subexponential-time algorithms [10,28]. The first protocol enjoying both concurrent zero-knowledge and concurrent soundness under standard complexity assumptions was presented in [12]. Based on this protocol, [27,6] proposed similar protocols with improved efficiency properties, under number-theoretic assumptions: the Decision Diffie-Hellman and the Discrete Logarithm problem, respectively.

Our results. Under the assumption of the existence of any one-way function family, we show that every language in NP has a 4-message argument system in the bare public key (BPK) model of [3], which satisfies concurrent soundness and (black-box) concurrent zero-knowledge. We stress the optimality properties of this protocol: the existence of one-way function families is a necessary assumption [24], and 4 messages are optimal among black-box zero-knowledge protocols unless NP is in BPP [22]. In comparison, the closest previous result [12], obtained such a protocol using a stronger assumption (the existence of one-way permutations) or a higher round complexity (5 messages).

While proving the zero-knowledge property of our protocol, we describe a non-trivial variant of a simulation strategy from [17] (for sequential zero-knowledge proofs in the standard model) and apply it to concurrent zero-knowledge arguments in the BPK model. Applications of our main technique (a novel use of the OR-based paradigm from [16]) include minimizing round complexity and complexity assumptions in concurrent non-malleable zero-knowledge arguments in the BPK model [7], arguably a more appealing property for zero-knowledge protocols on the Internet. Finally, we also discuss properties and variants of our protocol, related to arguments of knowledge, and achieving perfect zero-knowledge, and time and communication efficient protocols.

2 Definitions

We recall known definitions of the BPK model, and two main cryptographic tools that we will use in our constructions.

Model description. The BPK model can be seen as a relaxed version of two known models in Cryptography: the public-key infrastructure model, and the preprocessing model. Formally, the BPK model assumes that: (1) there exists a public file F that

is a collection of records, each containing a public key; (2) an (honest) prover is an interactive deterministic polynomial-time Turing machine that takes as input a security parameter 1^n , F , an n -bit string x , such that $x \in L$, for some language L , an auxiliary input y , a reference to an entry of F and a random tape; (3) an (honest) verifier V is an interactive deterministic polynomial-time Turing machine that works in the following two stages: on input a security parameter 1^n and a random tape, V generates a key pair (pk, sk) and stores the public key pk in one entry of the file F , and later, V takes as input the secret key sk , a statement $x \in L$ and a random string, and outputs “accept” or “reject” after performing an interactive protocol with a prover; (4) the first interaction between a prover and a verifier starts after all verifiers complete their first stage.

Malicious provers in the BPK model. Let p be a positive polynomial. We say that P^* is a p -concurrent malicious prover if it is a probabilistic polynomial-time Turing Machine that, on input 1^n and PK , can perform the $p(n)$ interactive protocols with V as follows: (1) if P^* is already running i protocols, $0 \leq i < p(n)$, he can choose a new statement x_i to be proved and start a new protocol with V with $x_i \in L$ as statement; (2) he can output a message for any running protocol, receive immediately the response from V and continue. (We assume that each message is unambiguously associated with only one of the protocols.) Given an s -concurrent malicious prover P^* and an honest verifier V , an s -concurrent attack is performed as follows: (1) the first stage of V is run on input 1^n and a random string to obtain pair (pk, sk) ; (2) P^* is run on input 1^n and pk so to obtain an n -bit string x_1 ; (3) whenever P^* starts a new protocol choosing an n -bit string x_i , V uses inputs x_i , a new random string r_i and sk , and interacts with P^* .

Malicious verifiers in the BPK model. In this paper we will consider concurrent zero-knowledge arguments. Let s be a positive polynomial. We say that V^* is an s -concurrent malicious verifier if it is probabilistic polynomial-time Turing Machine that, on input 1^n and PK , can perform the following $p(n)$ interactive protocols with P : (1) if V^* is already running $i - 1$ protocols $0 \leq i < p(n)$ he can decide the i -th protocol to be started with P ; (2) V^* can output a message for any running protocol, receive immediately the next message from P and continue. Given an s -concurrent malicious verifier V^* and an honest prover P , an s -concurrent attack is performed as follows: (1) in its first stage, V^* , on input 1^n and a random string, generates a public file F ; (2) V^* is run on input 1^n and F so to start the first protocol with P ; (3) whenever V^* starts a new protocol, P uses a new statement, a new random string, and interacts with V^* .

Definition 1. Given a language $L \in \text{NP}$, and its corresponding relation R_L , we say that a pair (P, V) satisfies *completeness* over L , if for all n -bit strings $x \in L$ and any witness y such that $(x, y) \in R_L$, the probability that V , at the end of the interaction with P on input y , outputs “reject”, is negligible in n . We say that (P, V) satisfies *concurrent soundness* over L , if, for any false statement “ $x \in L$ ”, for all positive polynomials p , for any p -concurrent prover P^* , the probability that in an execution of a p -concurrent attack V outputs “accept” for such a statement is negligible in n . Finally, we say that (P, V) satisfies *concurrent zero-knowledge* over L if for all positive polynomials p , for any p -concurrent verifier V^* , there exists a probabilistic polynomial-time algorithm

S_{V^*} , called the *simulator*, such that for all distinct $x_1, \dots, x_{p(n)} \in L$, the probability distributions $\{\text{view}_{V^*}^P(\bar{x})\}$ and $\{S_{V^*}(\bar{x})\}$ are computationally indistinguishable, where $\{\text{view}_{V^*}^P(\bar{x})\}$ is the distribution of the transcript seen by V^* on its input (i.e., $\bar{x} = x_1, \dots, x_{p(n)}$), random and communication tape while interacting with P .

Cryptographic Tools. We review two cryptographic primitives that are used in our main construction: secure signature schemes and commitment schemes.

Let n denote a security parameter. A *secure digital signature scheme* SS is a triple of (probabilistic) polynomial-time algorithms $\text{SS} = (\text{G}, \text{Sig}, \text{Ver})$ satisfying: *Correctness*: For all messages $m \in \{0, 1\}^k$, $\text{Prob}[(\text{pk}, \text{sk}) \leftarrow \text{G}(1^k); \hat{m} \leftarrow \text{Sig}(m, \text{pk}, \text{sk}) : \text{Ver}(m, \hat{m}, \text{pk}) = 1] = 1$. *Unforgeability*: For all probabilistic polynomial-time algorithms A , it holds that $\text{Prob}[(\text{pk}, \text{sk}) \leftarrow \text{G}(1^k); (m, \hat{m}) \leftarrow A^{\mathcal{O}(\text{pk}, \text{sk})}(\text{pk}) : m \notin \text{Query and Ver}(m, \hat{m}, \text{pk}) = 1]$ is negligible in n where $\mathcal{O}(\text{pk}, \text{sk})$ is a *signature oracle* that on input a message returns as output a signature of the message and Query is the set of messages for which A has requested a signature from $\mathcal{O}$. Secure signature schemes exist under the assumption of the existence of one-way functions secure against polynomial-time [26].

A *(2-message, computationally-hiding, statistically-binding) commitment scheme* is a pair of (probabilistic) polynomial-time algorithms (Com, Rec) satisfying: *Correctness*: For all $b \in \{0, 1\}$ and for all k , $\text{Prob}[s \leftarrow \text{Rec}(1^k); (\text{com}, \text{dec}) \leftarrow \text{Com}(1^n, b, s) : \text{Rec}(s, \text{com}, \text{dec}) = b] = 1$; *(Statistical) Binding*: For all n , and any string com , the probability (over the generation of s) there exists more than one bit $b \in \{0, 1\}$ such that $\text{Rec}(s, \text{com}, \text{dec}_b) = b$, for some string dec_b , is exponentially small in n . *(Computational) Hiding*: For any s , the distributions $\{[(\text{com}, \text{dec}) \leftarrow \text{Com}(1^n, 0, s) : \text{com}]\}_{n>0}$ and $\{[(\text{com}, \text{dec}) \leftarrow \text{Com}(1^n, 1, s) : \text{com}]\}_{n>0}$ are indistinguishable by polynomial-time algorithms. The above definition can be easily extended to the case in which we wish to commit to a string (instead of committing to a bit). Such commitment scheme exists under the assumption that there exist pseudo-random generators [20], which, in turn, exist under the existence of one-way function families [19].

3 Our Main Result

We present our main result by stating the formal theorem, and describing the protocol and its properties.

Theorem 1. Let L be a language in NP. Assuming the existence of a one-way function family, there exists (constructively) a 4-message argument system that satisfies completeness, concurrent soundness, and concurrent zero-knowledge over L .

Remarks on optimality properties of our protocol. The minimality of the complexity assumption follows from [24]. The round-optimality among black-box zero-knowledge protocols in the BPK model follows from [22]. Furthermore, when using complexity assumptions against polynomial-time algorithms, the level of soundness or zero-knowledge achieved by our protocol seem the best possible given that all currently known techniques to achieve improved soundness or zero-knowledge levels

(such as resettable or even bounded-resettable soundness or black-box zero-knowledge) in the BPK model, require complexity assumptions against subexponential-time algorithms.

An informal description of our protocol. Our argument system builds on the very often used ‘OR-based paradigm’, first introduced by [16], and adapted to our setting as follows. The prover proves to the verifier that either the original statement is true or so is some other statement, obtained from the transcript τ of the communication so far. Specifically, the prover creates a certain NP statement st_τ having the following two properties: 1) if τ is honestly generated by prover and verifier, then with high probability st_τ is false; 2) for any probabilistic polynomial-time verifier, there exists an efficient simulator that generates a transcript τ which is computationally indistinguishable from the analogue transcript generated during a real execution of the protocol between prover and this verifier, and such that st_τ is true. Then, in order to prove the NP statement input ‘ $x \in L$ ’, a prover will rather prove the statement ‘ $(x \in L) \vee st_\tau$ ’.

A technique often employed in almost all zero-knowledge protocols in the BPK model (first proposed in [3]) is that of constructing st_τ as the statement that there exists a secret key matching with the verifier’s public key in the public file, and then requiring the verifier to give a proof of knowledge of such secret key. Unfortunately, using this technique would result in a 5-message protocol or a 4-message protocol based under stronger than minimal complexity assumptions. Instead, we replace this technique by using signature schemes: the verifier publishes the verification key of a secure signature scheme in the public file, and uses the matching secret key to provide signatures during the protocol. In results from [22,11,9] this technique is used for the verifier to sign the common input and/or messages sent by the prover. Using these ideas, we could define st_τ as the statement saying that there exists a signature of the prover’s message, but it turns out that this would not be sufficient to achieve concurrent soundness or zero-knowledge. Our first idea is requiring the verifier to sign an unique session identifier, built as the concatenation of an n -bit random message m_v sent by the verifier and an n -bit random message m_p sent by the prover. Note that such signature of $m_v|m_p$ is especially tied to this session, as, in a different concurrent session, the verifier will independently generate a string m'_v , and, for any m'_p , a signature of $m'_v|m'_p$ cannot be used as a valid signature in the previous session. (This would guarantee concurrent soundness.) Instead, a simulator can rewind the verifier and obtain 2 signatures of concatenations $m_v|m_{p,i}$, for $i = 1, 2$, and therefore obtain a witness for st_τ to complete the simulation. (This might seem enough to guarantee some form of zero-knowledge.)

In fact, several attempts in proving concurrent zero-knowledge were not successful. Specifically, assume that for one session i , a simulator has obtained 2 valid signatures for distinct messages with all prefixes equal to the verifier’s string, and assume that session i is entirely contained into session j ; it could happen that a rewinding of session j demanded by the simulator may result in a complete rewinding of session i , and the work performed to simulate i will go lost. This problem is similar to the basic problem in proving that known constant-round sequential zero-knowledge protocols are also concurrent zero-knowledge, as discussed in [15]. We get around this problem with the following minor but crucial modification: instead of requiring that the simulator obtains 2 valid signatures for distinct messages with *all prefixes equal to the verifier’s random*

string, we only require that the simulator obtains 2 valid signatures for distinct messages with an *equal random prefix*. With respect to our previous informal discussion, this implies that the work done by the simulator with respect to a given session is never lost, and thus concurrent zero-knowledge seems guaranteed.

The above discussion assumed that a cheating verifier always returns a valid message while running its protocol. Of course, this is not always the case, and, in the case of constant-round/message protocols in the standard model, this complicates the proof of the zero-knowledge property, as first shown in [17]. Here, a simulation strategy was provided to deal with this specific adversary behavior, and to prove that their protocol is a constant-round zero-knowledge proof system for all languages in NP in the standard model. While proving the zero-knowledge property of our protocol, we run into a similar problem, but we need to prove a stronger property (concurrent zero-knowledge) in a less general model (the BPK model). Later, we provide a non-trivial modification to the simulation technique from [17] so that it applies to our protocol.

Formal description. Embedded in statement st_τ is an auxiliary language T_L that depends on L and on messages sent during the protocol. We start our formal description by formally defining T_L , then listing the tools and assumptions behind our protocol, and finally give a formal description of our protocol and its properties.

Definition 2. The 4-tuple $x(T_L) = (x, s, com, pk)$ belongs to the language T_L if $x \in L$ or there exist $w(T_L) = (m_v, m_{p,1}, m_{p,2}, sig_1, sig_2, dec)$ such that

1. $m_{p,1} \neq m_{p,2}$;
2. $Ver(m_v | m_{p,i}, sig_i, pk) = 1$ for $i = 1, 2$, and
3. $Rec(s, com, dec) = sig_1 | sig_2$.

We use $\ell_x(n), \ell_w(n)$ to denote the lengths $|x(T_L)|, |w(T_L)|$, respectively.

Informally speaking, 4-tuple (x, s, com, pk) belongs to T_L if x belongs to L or if com is the commitment of 2 valid signatures sig_1, sig_2 (with respect to pk) of two $2n$ -bit strings with the same n -bit prefix. In our construction we assume the existence of the following cryptographic tools.

1. a secure signature scheme $SS = (G, Sig, Ver)$, as from Section 2;
2. a 2-message, computationally-hiding, and statistically-binding commitment scheme (Com, Rec) , as from Section 2;
3. a 4-message public-coin witness-indistinguishable argument (kP, kV) of knowledge for the polynomial-time relation $R(T_L)$ associated with language $T_L \in NP$. We denote by (set, com, ch, ans) the 4 messages exchanged by prover kP and verifier kV . Then we assume that (kP, kV) satisfies a *special witness-extraction* property (an extractor, given two accepting transcripts $(set, cmes, ch, ames)$ and $(set, cmes, ch', ames')$ with $ch' \neq ch$, can compute a witness $w(T_L)$ for the common input $x(T_L)$ such that $(x(T_L), w(T_L)) \in R(T_L)$).

As discussed in Section 2, the first two tools in the above list can be constructed assuming any one-way function family, thanks to the results in [26,20,19]. We can obtain the protocol in item 3 from any one-way function family, as follows: we reduce T_L to an NP-complete language L' and start from the 3-round public-coin honest-verifier

zero-knowledge proof of knowledge π_{np} for relation $R(L')$ from [1] based on any 1-message commitment scheme, used by the prover, and we replace this commitment scheme with a 2-message commitment scheme based on any one-way function [20,19].

In Figure 1 we give a formal description of our protocol. Proving the completeness property is, as usual, simple, and we can prove the soundness property by combining techniques from [9,12]. Here, we discuss the (perhaps more interesting) proof for the concurrent zero-knowledge property.

KEY GENERATION PHASE.

V: On input a security parameter k , run the key generator G for the σ -secure signature scheme on input 1^k obtaining the pair (spk, ssk) . Let $pk = spk$ and $sk = ssk$.

PROOF PHASE.

Common input: security parameter k , public file $F = \{(pk, \dots)\}$ and instance x .

P's private input: a witness y for $x \in L$. **V's private input:** private key sk .

V(message 1): randomly choose $m_v \in \{0, 1\}^n$, compute $s = \text{Rec}(1^n)$, $set = kV(1^{\ell_x(n)})$, and send m_v, s, set to P ;

P(message 2): randomly choose $m_p \in \{0, 1\}^n$, $u \in \{0, 1\}^{\ell_w(n)}$; compute $(com, dec) = \text{Com}(1^n, u, s)$, $cmes = kP(1^{\ell_x(n)}, set)$; and send $m_p, com, cmes$ to V ;

V(message 3): compute $sig = \text{Sig}(m_v|m_p, sk)$ and $ch = kV(x(T_L), set, cmes)$, and send (sig, ch) to P ;

P(message 4): if $\text{Ver}(m, sig, pk) = 1$ then compute $ames = kP(x(T_L), set, cmes, ch)$ and send $ames$ to V ;

V(decision): if $kV(x(T_L), set, cmes, ch, ames) = \text{accept}$ then return: *accept* else return: *reject*.

Fig. 1. Our concurrently sound and zero-knowledge argument for NP in the BPK model

Concurrent zero-knowledge. Recall that the simulator S_{V^*} does not have any witness for any of the statements $x_i \in L$, and thus cannot run the prover's algorithm. However, it can rewind the verifier, and use this additional power to complete the simulation in expected polynomial time. As is often the case in the presence of concurrent attacks, we need to take special care to ensure that the number of rewindings required by S_{V^*} to complete the simulation remains polynomial on average. Informally speaking, the simulator S_{V^*} could be constructed using the following rewinding-based approach. Overall, the simulator S_{V^*} tries to use rewinding of V^* to compute two signatures sig_1, sig_2 of $m_v|m_{p,1}, m_v|m_{p,2}$, respectively, (i.e., the witness $w(T_L)$). This is achieved as follows: first S_{V^*} obtains a signature from V^* of a first message $(m_v|m_{p,1})$ at the end of the second message from V^* ; then, S_{V^*} rewinds V^* until the end of the first message from V^* , and sends a message containing a new and independently chosen n -bit string $m_{p,2}$, thus asking V^* to generate a signature for message $(m_v|m_{p,2})$. From sig_1, sig_2 , S_{V^*} could complete the simulation of the given session, without any further rewinding, by just committing to $w(T_L)$ in its first message. Note that if V^* sends a correct signature

when facing this different commitment, the simulation of this session is completed, and so is the entire simulation task, since there can be at most a polynomial number of sessions. However, the main problem is, as in [17], that V^* may send a correct signature after receiving a commitment to a random string, and an incorrect one after receiving a commitment to $w(T_L)$. In this situation, if S_{V^*} would return the conversation obtained so far and halt, then the simulation would be easily distinguishable from the real execution of the protocol. On the other hand, this would not be the case if S_{V^*} would repeatedly rewind V^* until a valid signature is obtained. However, the expected running time of this approach would be p_2/p_1 , where p_1 (resp., p_2) is the probability that V^* sends a valid signature after receiving a commitment to a random string (resp., to $w(T_L)$), and the computational binding property of the commitment scheme, implying that $|p_1 - p_2|$ is negligible, would not suffice to imply that p_2/p_1 is a polynomial. The approach used in [17] to bypass this problem can be described, in the context of our protocol, as follows: first, S_{V^*} tries to compute an estimate $\bar{p}_1$ of p_1 and then repeatedly rewinds V^* , up to $\text{poly}(n)/\bar{p}_1$ times, until it obtains the desired second signature. In [17] the authors show that this simulation is successful, except with negligible probability, and that the expected running time is polynomial. However, [17] proved sequential zero-knowledge in the standard interactive protocol model. Instead, here we need to prove a stronger requirement, concurrent zero-knowledge, in a less general setting, the BPK model.

Our simulation approach. In the presence of multiple concurrent sessions, one could apply to each session the single-session simulation strategy from [17]. This would work if sessions are not interleaved, but one of the main features of concurrent attacks is precisely to interleave sessions. Here, we face a simulation problem similar as before: rewinding V^* until after the first message of a session (say, while estimating probability p_1 for that session or while attempting to obtain a valid signature from V^*) might rewind all messages of another session, for which the previous rewinding work would go lost. The main idea in bypassing this problem is to change the rewinding rule. First, we say that the session associated with the received message is in one of these four phases:

1. *first signature phase:* S_{V^*} is attempting to obtain a valid first signature for a message of the kind $(m_v|m_p)$;
2. *probability estimation phase:* S_{V^*} has obtained a valid first signature and is now attempting to estimate the probability p_1 that V^* returns a valid first signature for a message of the kind $(m_v|m_p)$ after receiving a commitment to a random string;
3. *valid reply phase:* S_{V^*} has obtained an estimate of p_1 , as well as value $w(T_L)$ and is now attempting to obtain a valid reply from V^* after committing to $w(T_L)$ rather than to a random string;
4. *solved session phase:* S_{V^*} obtained an estimate of p_1 and a valid reply from V^* after committing to $w(T_L)$, and can use $w(T_L)$ to simulate the final message.

Given the above definitions, the rewinding rule is changed as follows: Instead of rewinding V^* until after the first message of the same session, our simulator rewinds V^* until after the first message of the previous session which is in a probability estimation phase or in a valid signature phase, and continues the simulation of that session. If no such session exists, then the simulator rewinds the same session. Note that this rewinding strategy will never waste any previous simulation work. By using a combinatorial game

analogy, we can consider each session as having a counter that needs to go from 0 (rewinding start time, when $r_1(n)$ rewindings are needed to conclude the probability estimation phase, and $r_2(n)$ rewindings are needed to conclude the valid reply phase) to $r_1(n) + r_2(n)$ (rewinding end time, when no more rewinding is needed to conclude the probability estimation phase or the valid reply phase). Now, note that the original rewinding strategy was incrementing counters of the rewind session but potentially decrementing counters in other, interleaved, sessions. In principle, convergence of the rewinding approach could not be guaranteed. Instead, the new rewinding strategy increments counters of either the rewind session or the previous session which is in a probability estimation phase or in a valid signature phase. This approach allows us to reuse much of the analysis from [17], and we can both show that the simulator's output is computationally indistinguishable from a real protocol execution, and that the simulator's expected running time is polynomial.

4 Extensions and Applications of Our Main Protocol

We discuss various properties, extension and applications of our main protocol.

Perfect zero-knowledge arguments. Under the assumption of the existence of any 2-message perfectly-hiding computationally-binding commitment scheme, we can show a 4-message (black-box) perfect zero-knowledge argument system for any language in NP. This is achieved by starting from our main protocol, and then replacing the commitment scheme used with the one claimed in the assumption. (As such schemes exist under any collision-intractable hash functions, this improves the result from [6], based on number-theoretic assumptions.)

Arguments of knowledge. Similarly as done in [13] for the protocol in [12], we can show that our main protocol is an argument of knowledge, in the BPK model, for the polynomial-time relation associated to language L .

Concurrently non-malleable zero-knowledge. Under the assumption of the existence of any one-way function family, we can show a 4-message concurrently non-malleable (black-box) zero-knowledge argument of knowledge for any polynomial-time relation in the BPK model. This is achieved by starting from the protocol in [7], and then replacing the verifier's (sub)proof with our signature-based technique and then use our novel variant of the OR-based paradigm.

Efficiency for Σ protocols. Let L be a language in NP, let U be the language of tuples (s, com, pk) such that com is a commitment to a signature verifiable using pk (formally, U can be defined in a way similar and simpler than T_L), and let $R(L), R(U)$ be their associated polynomial-time relations, respectively. Now, assume that there exists a Σ protocol [5] for both relations $R(L), R(U)$. We obtain that there exists a 4-message concurrently sound and zero-knowledge argument system for L with efficient time and communication complexity. Specifically, the communication complexity is $O(n)$, where n is the input length, and both prover and verifier only need to compute $O(1)$ modular exponentiations. This is achieved with the following minor modifications to protocol

(P,V). First, we define language T_L so that instead of referring to a single commitment com , to the tuple $(m_v, m_{p,1}, m_{p,2}, sig_1, sig_2, dec)$, refers to two commitments com_i , to the tuples $(m_v, m_{p,i}, sig_i, dec_i)$, for $i = 1, 2$. Second, we implement the proof of knowledge (kP, kV) for relation $R(T_L)$, by properly combining via the OR and AND techniques from [5,8], the two assumed Σ protocols for $R(U)$ and $R(L)$. We remark that a Σ protocol for $R(U)$ was given in [2], based on the strong RSA assumption.

References

1. Blum, M.: How to Prove a Theorem So No One Else Can Claim It. In: Proc. of ICM 1986 (1986)
2. Camenisch, J.L., Lysyanskaya, A.: A Signature Scheme with Efficient Protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
3. Canetti, R., Goldreich, O., Goldwasser, S., Micali, S.: Resettable Zero-Knowledge. In: Proc. of the 32nd ACM STOC (2000)
4. Canetti, R., Kilian, J., Petrank, E., Rosen, A.: Black-Box Concurrent Zero-Knowledge Requires $\omega(\log n)$ Rounds. In: Proc. of the 33rd ACM STOC (2001)
5. Cramer, R., Damgård, I.B., Schoenmakers, B.: Proof of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
6. Deng, Y., Lin, D.: Efficient Concurrent Zero Knowledge Arguments for NP in the Bare Public-Key Model. Journal of Software 19(2) (2008)
7. Deng, Y., Di Crescenzo, G., Lin, D., Feng, D.: Concurrently Non-Malleable Black-Box Zero Knowledge in the Bare Public-Key Model. In: CSR 2009. LNCS, vol. 5675. Springer, Heidelberg (2009)
8. De Santis, A., Di Crescenzo, G., Persiano, G., Yung, M.: On Monotone Formula Closure of SZK. In: Proc. of IEEE FOCS (1994)
9. Di Crescenzo, G., Lipmaa, H.: 3-message NP Argument in the BPK Model with Optimal Soundness and Zero Knowledge. In: ISAAC 2008. LNCS, vol. 5369, Springer, Heidelberg (2008)
10. Di Crescenzo, G., Persiano, G., Visconti, I.: Constant-round resettable zero knowledge with concurrent soundness in the bare public-key model. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 237–253. Springer, Heidelberg (2004)
11. Di Crescenzo, G., Persiano, G., Visconti, I.: Improved Setup Assumptions for 3-Round Resettable Zero Knowledge. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 530–544. Springer, Heidelberg (2004)
12. Di Crescenzo, G., Visconti, I.: Concurrent Zero Knowledge in the Public-Key Model. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 816–827. Springer, Heidelberg (2005)
13. Di Crescenzo, G., Visconti, I.: On Defining Proofs of Knowledge in the Public-Key Model. In: Proc. of ICTCS 2007, World Scientific, Singapore (2007)
14. Dwork, C., Naor, M.: Zaps and their applications. In: Proc. of 41st IEEE FOCS (2000)
15. Dwork, C., Naor, M., Sahai, A.: Concurrent Zero-Knowledge. In: Proc. of 30th ACM STOC (1998)
16. Feige, U., Lapidot, D., Shamir, A.: Multiple Non-Interactive Zero Knowledge Proofs Under General Assumptions. SIAM J. on Computing 29 (1999)
17. Goldreich, O., Kahan, A.: How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. J. Cryptology 9(3), 167–190 (1996)

18. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof-Systems. *SIAM J. on Computing* 18 (1989)
19. Hastad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. *SIAM Journal of Computing* 28 (1999)
20. Naor, M.: Bit Commitment Using Pseudo-Randomness. *J. of Cryptology* 4, 151–158 (1991)
21. Richardson, R., Kilian, J.: On the Concurrent Composition of Zero-Knowledge Proofs. In: Stern, J. (ed.) *EUROCRYPT 1999*. LNCS, vol. 1592, pp. 415–431. Springer, Heidelberg (1999)
22. Micali, S., Reyzin, L.: Soundness in the Public-Key Model. In: Kilian, J. (ed.) *CRYPTO 2001*. LNCS, vol. 2139, pp. 542–565. Springer, Heidelberg (2001)
23. Micali, S., Reyzin, L.: Min-round Resettable Zero-Knowledge in the Public-Key Model. In: Pfitzmann, B. (ed.) *EUROCRYPT 2001*. LNCS, vol. 2045, pp. 373–393. Springer, Heidelberg (2001)
24. Ostrovsky, R., Wigderson, A.: One-way Functions are Essential for Non-Trivial Zero-Knowledge. In: *Proc. 2nd ISTCS 1993*, IEEE Computer Society Press, Los Alamitos (1993)
25. Prabhakaran, M., Rosen, A., Sahai, A.: Concurrent Zero-Knowledge with Logarithmic Round Complexity. In: *Proc. of 43rd IEEE FOCS* (2002)
26. Rompel, J.: One-Way Functions are Necessary and Sufficient for Digital Signatures. In: *Proc. of the 22nd ACM STOC* (1990)
27. Visconti, I.: Efficient Zero Knowledge on the Internet. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) *ICALP 2006*. LNCS, vol. 4052, pp. 22–33. Springer, Heidelberg (2006)
28. Yung, M., Zhao, Y.: Generic and Practical Resettable Zero-Knowledge in the Bare Public-Key Model. In: Naor, M. (ed.) *EUROCRYPT 2007*. LNCS, vol. 4515, pp. 129–147. Springer, Heidelberg (2007)
29. Zhao, Y., Deng, X., Lee, C., Zhu, H.: Resettable Zero-Knowledge in the Weak Public-Key Model. In: Pfitzmann, B. (ed.) *EUROCRYPT 2001*. LNCS, vol. 2045, Springer, Heidelberg (2001)

Efficient Non-interactive Range Proof

Tsz Hon Yuen¹, Qiong Huang², Yi Mu¹, Willy Susilo¹,
Duncan S. Wong², and Guomin Yang²

¹ University of Wollongong, Australia
{thy738,ymu,wsusilo}@uow.edu.au

² City University of Hong Kong, China
{csqhuang@student.,duncan@,csyanggm@cs.}cityu.edu.hk

Abstract. We propose the first constant size non-interactive range proof which is not based on the heuristic Fiat-Shamir transformation and whose security does not rely on the random oracle assumption. The proof consists of a constant number of group elements. Compared with the most efficient constant-size range proof available in the literature, our scheme has significantly reduced the proof size. We showed that our scheme achieves perfect completeness, perfect soundness and composable zero-knowledge under a conventional number-theoretic assumption, namely the Subgroup Decision Problem.

1 Introduction

Proving in zero-knowledge that a committed value lies within a specified integer range is called *range proof*. Consider the following scenario: suppose that there is a firewall which grants the access of some private network only to users from a specific range of IP addresses, say with the same class A IP prefix “10.*.*”. Each user when accessing the private network has to prove to the firewall that he has a valid credential corresponding to his IP address, while he does not want to reveal his actual IP address to the firewall due to some privacy concern. Suppose that the user’s IP address is 10.168.0.1 and he is holding an anonymous credential for his corresponding IP value $178782209 = 10 \times 256^3 + 168 \times 256^2 + 1$. The user can prove to the firewall, using the *range proof*, that his IP value lies in the range $[10 \times 256^3, 10 \times 256^3 + 255 \times 256^2 + 255 \times 256 + 255]$, without revealing exactly what his IP address is.

Range proof has many other applications. For example, in some anonymous credential system [20] and e-cash system, [7], a prover can show that he is old enough (e.g. age ≥ 18) to access some sensible information; or show that the sequence number of an e-cash lies within a specified range, respectively.

In the literature, there are a number of range proof schemes available [5, 9, 18, 4, 17, 12, 6]. Most of the schemes are interactive and have to use the Fiat-Shamir transformation [13] for converting to their non-interactive versions. The security of the transformation relies on the random oracle [1] assumption, which is considered to be heuristic. The security may not preserve when the random oracle is replaced by a hash function, even if the security of a scheme is

reduced to some complexity (or number-theoretic) assumptions [8]. In addition to this, each of the schemes has a number of additional limitations (see Sec. 1.2 for details). Some of them may be inefficient: the proof size is linear to that of either some desirable security level ([5]) or the range ([18, 12, 6]). At the security aspect, some schemes do not achieve perfect completeness ([9, 4, 12]), perfect soundness ([18, 5, 9, 4, 17, 12, 6]), or only have statistical zero-knowledge ([9, 4, 17, 6]).

A natural question which remains unanswered is whether it is possible to build a non-interactive range proof scheme which does not rely on the random oracle assumption, while at the same time, is efficient (i.e. constant size) and achieves perfect completeness, perfect soundness and desirably a stronger notion of zero-knowledge.

1.1 Our Results and Techniques Used

In this paper, we answer this question affirmatively, by proposing a constant size non-interactive range proof scheme. The scheme is not based on the Fiat-Shamir transformation and its security does not rely on the random oracle assumption. To the best of our knowledge, our scheme is the first constant size non-interactive range proof without relying on the random oracle assumption. In addition to this, the proof contains a constant number of group elements and is more efficient than all the comparable schemes. On the security, the scheme achieves perfect completeness, perfect soundness and by far, one of the strongest zero-knowledge notions, namely the composable zero-knowledge (which implies unbounded zero-knowledge) [15].

Regarding the techniques used in our constructions, we have borrowed ideas from some of the previous range proof schemes and also made use of some techniques from other types of zero-knowledge proof systems, but putting them together in an interesting way for achieving those desirable properties mentioned above. In particular, our scheme follows the typical approach for range proof: suppose a prover P wants to prove that a committed secret μ is in some integer interval $[a, b]$. P will show that both $\mu - a$ and $b - \mu$ are non-negative. To do so, the scheme first applies the classic Lagrange's theorem that any positive integer can be written as the sum of four squares, which can be found using the Rabin-Shallit algorithm [19]. Then, we borrow some of the techniques from Groth and Sahai's non-interactive witness-indistinguishable (NIWI) proof for bilinear groups [16] and turn it into the final non-interactive range proof scheme. The security proof of our scheme is done in the traditional common reference string model. The number theoretic assumption that our scheme relies on is the Subgroup Decision Problem, which was first proposed by Boneh, Goh and Nissim [3].

1.2 Related Work

Below is a brief review of the related range proof schemes. In the description, we use P and V to denote a prover and a verifier, respectively. If P proves that a

committed secret μ falls in an interval I while V is convinced that μ belongs to an interval $J \supseteq I$, then we say that the *expansion rate* of the range proof scheme is $|J|/|I|$.

In 1987, Brickell *et al.* [5] proposed the BCDG range proof that prove a committed secret lying in $[0, B]$ for some positive integer B . The scheme has perfect completeness and perfect zero-knowledge. It achieves “computational” soundness, that is, the probability that a cheating prover can succeed is bounded by 2^{-t} where t is the number of times that the proof is iterated. The expansion rate of the scheme is three and the proof size is proportional to the value of t .

In 1998, Chan, Frankel and Tsiounis [9] improved the BCDG range proof. We call this scheme as CFT range proof. CFT range proof achieves “computational” completeness with probability greater than $1 - 2^l$ for some security parameter $l \in \mathbb{N}$. The soundness achieved is also computational, where a cheating prover can succeed with probability bounded by 2^{-t} . Regarding zero-knowledge, the scheme achieves honest-verifier statistic zero-knowledge (HVSZK). The expansion rate is 2^{t+l+1} . The CFT range proof improves the BCDG range proof by having the proof size independent of the value of t .

With similar range structure to that of BCDG and CFT range proofs, Mao [18] proposed a range proof in 1998 for ranges in the form of $[0, 2^k - 1]$ where $k \in \mathbb{N}$. Mao’s proof size is proportional to the size of the range. The scheme has perfect completeness, perfect zero-knowledge, and “computational” soundness.

The first range proof scheme for the general range $[a, b]$ was proposed by Boudot in 2000 [4]. In addition, its expansion rate is exactly one. In other words, it is the first range proof which solves the expansion rate problem. The scheme has “computational” completeness and soundness, and the level of HVSZK with respect to zero-knowledge.

In 2003, Lipmaa [17] propose a range proof for committed secrets lying in $[0, B]$ where B is some positive integer. In Lipmaa’s proof, the following classic Lagrange’s theorem from the year 1770 was employed.

Theorem 1. *For an integer μ , there exist (efficiently computable) integers $\omega_1, \omega_2, \omega_3, \omega_4$ such that $\mu = \omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2$, if and only if $\mu \geq 0$.*

For finding the four squares in the theorem, Rabin and Shallit’s algorithm [19] can be used. Lipmaa’s scheme has perfect completeness, while the soundness is “computational” and the zero-knowledge is HVSZK.

Di Crescenzo, Herranz and Sáez [12] proposed a non-interactive range proof without random oracles in 2004. We call this scheme as DHS range proof. The DHS range proof decomposes the committed number using Mao’s method, and uses the NIZK proof for Blum integers and quadratic residue [11]. The DHS range proof has perfect zero-knowledge, while the completeness and the soundness are “computational”. The disadvantage of this scheme is that the communication complexity is proportional to the size of the range.

Groth [14] proposed a variation of Lipmaa’s method in 2005. It is based on an observation from number theory that the only numbers that cannot be written as the sum of three squares are of the form $4^m(8k + 7)$ for some integers m and

k . Specifically, $4\mu + 1$ can be written as a sum of three squares, which implies μ to be non-negative.

Recently in [6], Camenisch, Chaabouni and shelat proposed a new range proof. Their scheme is constructed from a set membership proof which is based on the Boneh-Boyen signature scheme [2]. This also implies that their scheme's security relies on the q -Strong Diffie-Hellman assumption, while the other range proofs are generally relying on the strong RSA assumption. Furthermore, their scheme has perfect completeness while having HVSZK in zero-knowledge and "computational" soundness. The proof size also depends on the size of the range.

2 Definitions and Number-Theoretic Assumption

2.1 Non-interactive Proof System

We review the definition for non-interactive proof based on the one given by Groth and Sahai recently in [16]. Let R be an efficiently computable ternary relation. For triplets $(gk, x, w) \in R$ we call gk the setup, x the statement and w the witness. Let L_R be the language such that

$$L_R(gk) \doteq \{x \mid (\exists w)[(gk, x, w) \in R]\}.$$

The standard definition of an NP-language often has gk omitted. In [16] and in this paper, gk is the description of a bilinear group.

A non-interactive proof system for R consists of four probabilistic polynomial time algorithms: a setup algorithm $\mathcal{G}$, a common reference string (CRS) generation algorithm K , a prover P and a verifier V . $\mathcal{G}$ takes as input the security parameter 1^k , and outputs a setup gk . It may also output some auxiliary information sk , for example, the factorization of a group order. Note that sk can simply be an empty string, meaning that the proof system is built upon a group without knowledge of any trapdoor. The CRS generation algorithm K takes (gk, sk) as input and produces a common reference string σ . P takes as input (gk, σ, x, w) and produces a proof π , while V takes as input (gk, σ, x, π) and outputs 1 for accepting the proof, or 0 for rejecting the proof. We call $(\mathcal{G}, K, P, V)$ a non-interactive proof system for R if it has the following properties.

Perfect Completeness. For all adversaries $\mathcal{A}$, we have

$$\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k); \sigma \leftarrow K(gk, sk); (x, w) \leftarrow \mathcal{A}(gk, \sigma); \pi \leftarrow P(gk, \sigma, x, w) : \\ V(gk, \sigma, x, \pi) = 1 \text{ if } (gk, x, w) \in R] = 1.$$

Perfect Soundness. For all adversaries $\mathcal{A}$, we have

$$\Pr[(gk, sk) \leftarrow \mathcal{G}(1^k); \sigma \leftarrow K(gk, sk); (x, \pi) \leftarrow \mathcal{A}(gk, \sigma) : \\ V(gk, \sigma, x, \pi) = 0 \text{ if } x \notin L_R(gk)] = 1.$$

Composable Zero-Knowledge.¹ For this notion of zero-knowledge, there are two aspects: first, an adversary should not be able to distinguish a real CRS from

¹ Composable zero-knowledge was first proposed by Groth [15]. In [15], Groth also showed that composable zero-knowledge implies unbounded zero-knowledge.

a simulated CRS; second, the adversary should not be able to distinguish real proofs on a simulated CRS from simulated proofs, even if he gets access to the secret simulation key τ . In other words, there exists a polynomial time simulator (S_1, S_2) that for all non-uniform polynomial time adversaries $\mathcal{A}$, we have

$$\begin{aligned} & \Pr[(gk, sk) \leftarrow \mathcal{G}(1^k); \sigma \leftarrow K(gk, sk) : \mathcal{A}(gk, \sigma) = 1] \\ & \approx \Pr[(gk, sk) \leftarrow \mathcal{G}(1^k); (\sigma, \tau) \leftarrow S_1(gk, sk) : \mathcal{A}(gk, \sigma) = 1], \end{aligned}$$

and

$$\begin{aligned} & \Pr[(gk, sk) \leftarrow \mathcal{G}(1^k); (\sigma, \tau) \leftarrow S_1(gk, sk); (x, w) \leftarrow \mathcal{A}(gk, \sigma, \tau); \\ & \quad \pi \leftarrow P(gk, \sigma, x, w) : \mathcal{A}(\pi) = 1] \\ & = \Pr[(gk, sk) \leftarrow \mathcal{G}(1^k); (\sigma, \tau) \leftarrow S_1(gk, sk); (x, w) \leftarrow \mathcal{A}(gk, \sigma, \tau); \\ & \quad \pi \leftarrow S_2(gk, \sigma, \tau, x) : \mathcal{A}(\pi) = 1], \end{aligned}$$

where $(gk, x, w) \in R$.

2.2 Pairing and Intractability Problem

Let $\mathbb{G}, \mathbb{G}_T$ be multiplicative groups of order $n = pq$, where p and q are prime. Let g be the generator of $\mathbb{G}$. We denote $\mathbb{G}_q$ as the subgroup of $\mathbb{G}$ with order q .

Definition 1. A map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is called a pairing if, for all $g \in \mathbb{G}$ and $a, b \in \mathbb{Z}_n$, we have $\hat{e}(g^a, g^b) = \hat{e}(g, g)^{ab}$, and if g is the generator of $\mathbb{G}$, then $\hat{e}(g, g)$ generates $\mathbb{G}_T$.

Definition 2 (Subgroup Decision Problem). Given $(n, \mathbb{G}, \mathbb{G}_T, \hat{e})$ and a random element $u \in \mathbb{G}$, output ‘1’ if the order of u is q and output ‘0’ otherwise.

The advantage of an algorithm $\mathcal{A}$ in solving the problem is defined as:

$$\begin{aligned} Adv_{\mathcal{A}}(k) = & \left| \Pr[\mathcal{A}(n, \mathbb{G}, \mathbb{G}_T, \hat{e}, u) = 1 : \begin{array}{l} (p, q, \mathbb{G}, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^k) \\ n = pq, u \leftarrow \mathbb{G}. \end{array} \right. \\ & \left. - \Pr[\mathcal{A}(n, \mathbb{G}, \mathbb{G}_T, \hat{e}, u) = 1 : \begin{array}{l} (p, q, \mathbb{G}, \mathbb{G}_T, \hat{e}) \leftarrow \mathcal{G}(1^k) \\ n = pq, u \leftarrow \mathbb{G}_q. \end{array} \right] \big|. \end{aligned}$$

The subgroup decision assumption assumes that for any polynomial time algorithm $\mathcal{A}$, $Adv_{\mathcal{A}}(k)$ is a negligible function in k . This assumption was first proposed by Boneh, Goh and Nissim [3].

3 Our Range Proof Scheme

As mentioned in Sec. 1.1, our approach is to prove that both $\mu_1 = \mu - a$ and $\mu_2 = b - \mu$ are non-negative for a committed secret μ which lies in $[a, b]$. To do this, P applies Theorem 1 and represents $\mu_i = \omega_{i,1}^2 + \omega_{i,2}^2 + \omega_{i,3}^2 + \omega_{i,4}^2$ for $i = 1, 2$,

using the Rabin and Shallit algorithm [19]. Then, P performs a proof for the statement

$$\mu_1 = \sum_{j=1}^4 \omega_{1,j}^2 \quad \wedge \quad \mu_2 = \sum_{j=1}^4 \omega_{2,j}^2. \quad (1)$$

using the witness $(\{\omega_{i,j}\}_{1 \leq i \leq 2; 1 \leq j \leq 4}, \mu)$. We borrow some of the techniques from Groth and Sahai's NIWI proof [16] and turn it into the final non-interactive range proof. In particular, we borrow the technique of commitment for pairings from [16] that is based on the subgroup decision assumption.

We follow the notations for pairings denoted in Sec. 2.2. Let u be either a generator of $\mathbb{G}$ or that of $\mathbb{G}_q$. We commit to $w \in \mathbb{Z}_p$, by choosing $\rho \in \mathbb{Z}_n$ at random and setting $T_w := g^w u^\rho$. If u 's order is q , w is uniquely determined in $\mathbb{Z}_p$, since $T_w^q = g^{wq}$; but if u 's order is n , then we have a perfectly hiding commitment to w . Under the subgroup decision assumption, the two types of commitments are computationally indistinguishable.

3.1 The Construction and Its Security

Suppose that $|b - a| \leq p$, the non-interactive range proof scheme is as follows:

- $\mathcal{G}(1^k)$: We define $\mathcal{G}$ such that it generates $gk = (n, \mathbb{G}, \mathbb{G}_T, \hat{e})$ and $sk = (p, q)$, where p and q are two random k -bit primes, $n = pq$, and $\hat{e}$ the pairing as described in Def. 1.
- $K(gk, sk)$: The CRS generation algorithm K takes as input $gk = (n, \mathbb{G}, \mathbb{G}_T, \hat{e})$ and $sk = (p, q)$, randomly selects a generator g of $\mathbb{G}$ and a generator u of $\mathbb{G}_q$, and outputs $\sigma = (g, u)$.
- $\mathsf{P}(gk, \sigma, x, w)$: The prover P takes as input $gk = (n, \mathbb{G}, \mathbb{G}_T, \hat{e})$, $\sigma = (g, u)$, x the statement (which is equation (1)) and w the witness of equation (1), and carries out the following: for $i = 1, 2, j = 1, \dots, 4$, P randomly chooses $r_{i,j} \in_R \mathbb{Z}_n$ and computes $T_{i,j} = g^{\omega_{i,j}} u^{r_{i,j}}$. Then P randomly picks $r_w \in_R \mathbb{Z}_n$ and computes $T_w = g^w u^{r_w}$. P calculates:

$$\begin{aligned} \phi_1 &= g^{-r_w + 2 \sum_{j=1}^4 r_{1,j} \omega_{1,j}} u^{\sum_{j=1}^4 r_{1,j}^2}, \\ \phi_2 &= g^{r_w + 2 \sum_{j=1}^4 r_{2,j} \omega_{2,j}} u^{\sum_{j=1}^4 r_{2,j}^2}. \end{aligned}$$

P sends the proof $\pi = (\{T_{1,j}, T_{2,j}\}_{j \in [4]}, T_w, \phi_1, \phi_2)$ to the verifier V .

- $\mathsf{V}(gk, \sigma, x, \pi)$: The verifier V takes as input $gk = (n, \mathbb{G}, \mathbb{G}_T, \hat{e})$, $\sigma = (g, u)$, x the statement (which is equation (1)) and $\pi = (\{T_{1,j}, T_{2,j}\}_{j \in [4]}, T_w, \phi_1, \phi_2)$ the proof, and checks if

$$\begin{aligned} \hat{e}(g^a T_w^{-1}, g) \cdot \prod_{j=1}^4 \hat{e}(T_{1,j}, T_{1,j}) &= e(u, \phi_1), \\ \hat{e}(T_w g^{-b}, g) \cdot \prod_{j=1}^4 \hat{e}(T_{2,j}, T_{2,j}) &= e(u, \phi_2). \end{aligned}$$

V outputs 1 if the equations hold; otherwise, outputs 0.

Our proof scheme described above can also be adapted to prove logical relations of ranges. Groth and Sahai [16] mentioned that logical operations like AND and OR are easy to encode into their framework using standard techniques in arithmetization. Therefore we can use our system to prove that $x \in [a, b] \vee x \in [c, d]$.

Theorem 2. *The range proof scheme described above is a non-interactive proof system satisfying perfect completeness, perfect soundness and composable zero-knowledge if the subgroup decision assumption holds.*

Proof. Perfect Completeness.

$$\begin{aligned}
& \hat{e}(g^a T_w^{-1}, g) \cdot \prod_{j=1}^4 \hat{e}(T_{1,j}, T_{1,j}) \\
&= \hat{e}(g^{a-w} u^{-r_w}, g) \cdot \prod_{j=1}^4 (\hat{e}(g, g)^{\omega_{1,j}^2} \cdot \hat{e}(u, g)^{2r_{1,j}\omega_{1,j}} \cdot \hat{e}(u, u^{r_{1,j}^2})) \\
&= \hat{e}(g, g)^{\sum_{j=1}^4 \omega_{1,j}^2 + a-w} \cdot \hat{e}(u, g^{-r_w + \sum_{j=1}^4 2r_{1,j}\omega_{1,j}} u^{\sum_{j=1}^4 r_{1,j}^2}) \\
&= \hat{e}(u, \phi_1), \\
& \hat{e}(T_w g^{-b}, g) \cdot \prod_{j=1}^4 \hat{e}(T_{2,j}, T_{2,j}) \\
&= \hat{e}(g^{w-b} u^{r_w}, g) \cdot \prod_{j=1}^4 (\hat{e}(g, g)^{\omega_{2,j}^2} \cdot \hat{e}(u, g)^{2r_{2,j}\omega_{2,j}} \cdot \hat{e}(u, u^{r_{2,j}^2})) \\
&= \hat{e}(g, g)^{\sum_{j=1}^4 \omega_{2,j}^2 + w-b} \cdot \hat{e}(u, g^{r_w + \sum_{j=1}^4 2r_{2,j}\omega_{2,j}} u^{\sum_{j=1}^4 r_{2,j}^2}) \\
&= \hat{e}(u, \phi_2).
\end{aligned}$$

Perfect Soundness. Notice that u is in $\mathbb{G}_q$. Therefore when we try to power q from both sides of the verification equations, the equations become:

$$\begin{aligned}
& \hat{e}(g^a T_w^{-1}, g)^q \cdot \prod_{j=1}^4 \hat{e}(T_{1,j}, T_{1,j})^q = (\hat{e}(g^{a-w}, g) \cdot \prod_{j=1}^4 \hat{e}(g, g)^{\omega_{1,j}^2})^q = 1, \\
& \hat{e}(T_w g^{-b}, g)^q \cdot \prod_{j=1}^4 \hat{e}(T_{2,j}, T_{2,j})^q = (\hat{e}(g^{w-b}, g) \cdot \prod_{j=1}^4 \hat{e}(g, g)^{\omega_{2,j}^2})^q = 1.
\end{aligned}$$

If $x \notin L$, then we have either $\mu_1 = \mu - a$ or $\mu_2 = b - \mu$ is negative. Theorem 1 states that if μ_i is negative, then we cannot represent μ_i as $\sum_{j=1}^4 \omega_{i,j}^2$. Therefore we have either $\mu - a \neq \sum_{j=1}^4 \omega_{1,j}^2$ or $b - \mu \neq \sum_{j=1}^4 \omega_{2,j}^2$. Therefore the proof cannot pass the verification.

Composable Zero-Knowledge. First, we prove that a PPT adversary cannot distinguish a real CRS from a simulated CRS. The simulated CRS is generated

as follows. Suppose the simulator S_1 takes as input $gk = (n, \mathbb{G}, \mathbb{G}_T, \hat{e})$ and $sk = (p, q)$. The simulator S_1 is also given u from the subgroup decision problem. S_1 randomly selects a generator g of $\mathbb{G}$. S_1 outputs the simulated CRS $\sigma = (g, u)$ and the secret simulation key $\tau = (p, q)$. If a PPT adversary $\mathcal{A}_1$ can distinguish a real CRS from σ , then S_1 answers 0 to the subgroup decision problem. By the subgroup decision assumption, no PPT adversary can distinguish a real CRS generated by K from a simulated CRS generated by S_1 .

Second, we prove that a PPT adversary cannot distinguish real proofs on a simulated CRS from simulated proofs, even if he gets access to the secret simulation key τ . Suppose the simulator S_2 takes as input $gk = (n, \mathbb{G}, \mathbb{G}_T, \hat{e})$, $\sigma = (g, u)$, $\tau = (p, q)$ and x is the proof statement. S_2 picks a random w_r from the range $[a, b]$. By theorem 1, S_2 can also represent $w_r - a$ and $b - w_r$ as sum of four squares. Therefore S_2 can calculate simulated commitments and proofs with this randomly chosen witness w_r .

Suppose there is a PPT adversary $\mathcal{A}_2$ can distinguish this simulated proof and the real proof with witness $(\{\omega_{1,j}, \omega_{2,j}\}_{j \in [4]}, w)$. The commitments $(\{T_{1,j}, T_{2,j}\}_{j \in [4]}, T_x)$ are perfect hiding if u is the generator of $\mathbb{G}$. They have the same distribution no matter we use the witness $(\{\omega_{1,j}, \omega_{2,j}\}_{j \in [4]}, w)$ or the randomly chosen witness $(\dots, w_r)$. Therefore $\mathcal{A}_2$ cannot distinguish from the commitment only.

Therefore $\mathcal{A}_2$ can only distinguish from the proofs (ϕ_1, ϕ_2) given the commitment. Theorem 3 of [16] tells us that the proofs (ϕ_1, ϕ_2) made with either one of the above witnesses are uniformly distributed over all possible choices of the corresponding domain. Contradiction occurs. $\square$

3.2 Comparison

We now compare our range proof with the existing ones found in the literature. During the comparison, we consider the security level comparable to 80-bit symmetric or 1024-bit RSA.

Suppose that the underlying pairing with composite order is constructed from a supersingular curve with embedding degree 2. The proof size (i.e. communication overhead) of our scheme is

$$11\mathbb{G} = 11 \times 1025 = 11275 \text{ bits.}$$

If we restrict the witness to have the form $4v + 1$ for some integer v , then we can have further reduce the proof size. For example, in the case of voting, we can represent candidate A as number 1, candidate B as number 5 and so on. Then we can represent the number of any candidate as the sum of three squares (Sec. 1.2) and the proof size will be reduced to $9\mathbb{G} = 9216$ bits, which is 18% less than the original one.

Recently, Camenisch, Chaabouni and shelat [6] proposed a range proof without relying on the strong RSA assumption. If the range is $< k - 1$ bits, then the proof size is $O(\frac{k}{\log k - \log \log k})$. Using the example in their paper, a prover wants to show that the committed secret lies between $[347184000, 599644800]$. Cheon [10] proved that the security of the ℓ -SDH problem on an abelian group

of order p can be reduced up to $O(\log p \cdot p^{1/3})$ (resp. $O(\log p \cdot p^{1/3})$) for large ℓ if $p - 1$ (resp. $p + 1$) has a divisor $d = O(p^{1/2})$ (resp. $d = O(p^{1/3})$). Because of this, Cheon suggested to use 220-bit prime p for 80-bit symmetric security level. Suppose that we use a pairing with embedding degree 6. The proof size² is

$$5G + 8G_T + 20Z_p = 5 \times 221 + 8 \times 1321 + 20 \times 220 = 16073 \text{ bits.}$$

Our scheme has the proof size 30% less than that of Camenisch *et al.*'s scheme. It is not hard to see that our scheme saves even more bandwidth if the range is larger.

It is more natural to compare our scheme with the existing constant size range proofs whose expansion rate equals to 1. We use the 1024-bit RSA security level for communication complexity comparison. The proof size of our scheme is about 42% less than that of existing schemes.

Table 1. Comparison of range proofs with expansion rate equals to 1. Complexity stands for the proof size. We omit the figure for Camenisch *et al.* [6] since it is not a constant size range proof. PC stands for perfect completeness. PS stands for perfect soundness. ZK stands for zero-knowledge. NI stands for non-interactive.

Scheme	Complexity	PC	PS	ZK	Assumption	Proof System
Boudot	23544 bits	×	×	HVSZK	strong RSA	interactive
Lipmaa	19536 bits	✓	×	HVSZK	strong RSA	interactive
Camenisch <i>et al.</i>	-	✓	×	HVSZK	ℓ -SDH	interactive
This paper	11275 bits	✓	✓	composable ZK	subgroup decision	NI

4 Conclusion

We proposed an efficient non-interactive range proof. To the best of our knowledge, this is the first constant size range proof which is not based on the Fiat-Shamir transformation and whose security does not rely on the random oracle assumption. The proof consists of constant number of group elements and is the most efficient range proof scheme in the literature. We showed that our scheme achieves perfect completeness, perfect soundness and composable zero-knowledge under the Subgroup Decision Problem.

References

1. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS 1993, pp. 62–73. ACM Press, New York (1993)
2. Boneh, D., Boyen, X.: Short Signatures Without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)

² Camenisch *et al.* [6] used 3072-bit RSA security level for comparison. However, they used a pairing of embedding degree 12 which has no known efficient implementation. They did not take into account of the attack on the ℓ -SDH problem by Cheon [10].

3. Boneh, D., Goh, E.-J., Nissim, K.: Evaluating 2-DNF Formulas on Ciphertexts. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 325–341. Springer, Heidelberg (2005)
4. Boudot, F.: Efficient Proofs that a Committed Number Lies in an Interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)
5. Brickell, E.F., Chaum, D., Damgård, I., van de Graaf, J.: Gradual and verifiable release of a secret. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 156–166. Springer, Heidelberg (1988)
6. Camenisch, J., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (2008)
7. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005)
8. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. *J. ACM* 51(4), 557–594 (2004)
9. Chan, A.H., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 561–575. Springer, Heidelberg (1998)
10. Cheon, J.H.: Security analysis of the strong diffie-hellman problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 1–11. Springer, Heidelberg (2006)
11. De Santis, A., Di Crescenzo, G., Persiano, G.: The knowledge complexity of quadratic residuosity languages. *Theor. Comput. Sci.* 132(2), 291–317 (1994)
12. Di Crescenzo, G., Herranz, J., Sáez, G.: Reducing server trust in private proxy auctions. In: Katsikas, S.K., López, J., Pernul, G. (eds.) TrustBus 2004. LNCS, vol. 3184, pp. 80–89. Springer, Heidelberg (2004)
13. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
14. Groth, J.: Non-interactive Zero-Knowledge Arguments for Voting. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 467–482. Springer, Heidelberg (2005)
15. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459. Springer, Heidelberg (2006)
16. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008)
17. Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: Lai, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)
18. Mao, W.: Guaranteed correct sharing of integer factorization with off-line shareholders. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 60–71. Springer, Heidelberg (1998)
19. Rabin, M., Shallit, J.: Randomized algorithms in number theory. *Communications in Pure and Applied Mathematics* 39, 239–256 (1986)
20. Teranishi, I., Furukawa, J., Sako, K.: k -times anonymous authentication (Extended abstract). In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 308–322. Springer, Heidelberg (2004)

Approximation Algorithms for Key Management in Secure Multicast

Agnes Chan¹, Rajmohan Rajaraman¹, Zhifeng Sun¹, and Feng Zhu²

¹ Northeastern University, Boston, MA 02115, USA

² Cisco Systems, San Jose, CA, USA

Abstract. Many data dissemination and publish-subscribe systems that guarantee the privacy and authenticity of the participants rely on symmetric key cryptography. An important problem in such a system is to maintain the shared group key as the group membership changes. We consider the problem of determining a key hierarchy that minimizes the average communication cost of an update, given update frequencies of the group members and an edge-weighted undirected graph that captures routing costs. We first present a polynomial-time approximation scheme for minimizing the average number of multicast messages needed for an update. We next show that when routing costs are considered, the problem is NP-hard even when the underlying routing network is a tree network or even when every group member has the same update frequency. Our main result is a polynomial time constant-factor approximation algorithm for the general case where the routing network is an arbitrary weighted graph and group members have nonuniform update frequencies.

1 Introduction

A number of data dissemination and publish-subscribe systems, such as interactive gaming, stock data distribution, and video conferencing, need to guarantee the privacy and authenticity of the participants. Many such systems rely on symmetric key cryptography, whereby all legitimate group members share a common key, henceforth referred to as the *group key*, for group communication. An important problem in such a system is to maintain the shared group key as the group membership changes. The main security requirement is *confidentiality*: only valid users should have access to the multicast data. In particular this means that any user should have access to the data only during the time periods that the user is a member of the group.

There have been several proposals for multicast key distribution for the Internet and ad hoc wireless networks [2,7,8,18,24]. A simple solution proposed in early Internet RFCs is to assign each user a *user key*; when there is a change in the membership, a new group key is selected and separately unicast to each of the users using their respective user keys [8,7]. A major drawback of such a key management scheme is its prohibitively high update cost in scenarios where member updates are frequent.

The focus of this paper is on a natural key management approach that uses a hierarchy of auxiliary keys to update the shared group key and maintain the desired security properties. Variations of this approach, commonly referred to as the *Key Graph* or the *Logical Key Hierarchy* scheme, were proposed by several independent groups of researchers [2,4,21,23,24]. The main idea is to have a single group key for data communication, and have a group controller (a special server) distribute auxiliary subgroup keys to the group members according to a key hierarchy. The leaves of the key hierarchy are the group members and every node of the tree (including the leaves) has an associated *auxiliary* key. The key associated with the root is the shared group key. Each member stores auxiliary keys corresponding to all the nodes in the path to the root in the hierarchy. When an update occurs, say at member u , then all the keys along the path from u to the root are rekeyed from the bottom up (that is, new auxiliary keys are selected for every node on the path). If a key at node v is rekeyed, the new key value is multicast to all the members in the subtree rooted at v using the keys associated with the children of v in the hierarchy.¹ It is not hard to see that the above key hierarchy approach, suitably implemented, yields an exponential reduction in the number of multicast messages needed on a member update, as compared to the scheme involving one auxiliary key per user.

The effectiveness of a particular key hierarchy depends on several factors including the organization of the members in the hierarchy, the routing costs in the underlying network that connects the members and the group controller, and the frequency with which individual members join or leave the group. Past research has focused on either the security properties of the key hierarchy scheme [3] or concentrated on minimizing either the total number of auxiliary keys updated or the total number of multicast messages [22], not taking into account the routing costs in the underlying communication network.

1.1 Our Contributions

In this paper, we consider the problem of designing key hierarchies that minimize the average update cost, given an arbitrary underlying routing network and given arbitrary update frequencies of the members, which we refer henceforth to as weights. Let S denote the set of all group members. For each member v , we are given a weight w_v representing the update probability at v (e.g., a join/leave action at v). Let G denote an edge-weighted undirected routing network that connects the group members with a group controller r . The cost of any multicast from r to any subset of S is determined by G . The cost of a given key hierarchy is then given by the weighted average, over the members v , of the sum of the costs of the multicasts performed when an update occurs at v . A formal problem definition is given in Section 2.

¹ We emphasize here that auxiliary keys in the key hierarchy are only used for maintaining the group key. Data communication within the group is conducted using the group key.

- We first consider the objective of minimizing the average number of multicast messages needed for an update, which is modeled by a routing tree where the multicast cost to every subset of the group is the same. For uniform multicast costs, we precisely characterize the optimal hierarchy when all the member weights are the same, and present a polynomial-time approximation scheme when member weights are nonuniform. These results appear in Section 3.
- We next show in Section 4 that the problem is NP-hard when multicast costs are nonuniform, even when the underlying routing network is a tree or when the member weights are uniform.
- Our main result is a constant-factor approximation algorithm in the general case of nonuniform member weights and nonuniform multicast costs captured by an arbitrary routing graph. We achieve a 75-approximation in general, and achieve improved constants of approximation for tree networks (11 for nonuniform weights and 4.2 for uniform weights). These results are in Section 5.

Our approximation algorithms are based on a simple divide-and-conquer framework that constructs “balanced” binary hierarchies by partitioning the routing graph using both the member weights and the routing costs. A key ingredient of our result for arbitrary routing graphs is the algorithm of [14] which, given any weighted graph, finds a spanning tree that simultaneously approximates the shortest path tree from a given node and the minimum spanning tree of the graph.

Due to space constraints, we have omitted many of the proofs in this paper. Please refer to the full version of the paper [5] for details.

1.2 Related Work

Variants of the key hierarchy scheme studied in this paper were proposed by several independent groups [2,4,21,23,24]. The particular model we have adopted matches the Key Graph scheme of [24], where they show that a balanced hierarchy achieves an upper bound of $O(\log n)$ on the number of multicast messages needed for any update in a group of n members. In [22], it is shown that $\Theta(\log n)$ messages are necessary for an update in the worst case, for a general class of key distribution schemes. Lower bounds on the amount of communication needed under constraints on the number of keys stored at a user are given in [3]. Information-theoretic bounds on the number of auxiliary keys that need to be updated given member update frequencies are given in [19].

In recent work, [16] and [20] have studied the design of key hierarchy schemes that take into account the underlying routing costs and energy consumption in an ad hoc wireless network. The results of [16,20], which consist of hardness proofs, heuristics, and simulation results, are closely tied to the wireless network model, relying on the broadcast nature of the medium. In this paper, we present approximation algorithms for a more basic routing cost model given by an undirected weighted graph.

The special case of uniform multicast costs (with nonuniform member weights) bears a strong resemblance to the Huffman encoding problem [11]. Indeed, it can be easily seen that an optimal *binary* hierarchy in this special case is given by

the Huffman code. The truly optimal hierarchy, however, may contain internal nodes of both degree 2 and degree 3, which contribute different costs, respectively, to the leaves. In this sense, the problem seems related to Huffman coding with unequal letter costs [12], for which a PTAS is given in [6]. The optimization problem that arises when multicast costs and member weights are both uniform also appears as a special case of the constrained set selection problem, formulated in the context of website design optimization [10]. Another related problem is broadcast tree scheduling where the goal is to determine a schedule for broadcasting a message from a source node to all the other nodes in a heterogeneous network where different nodes may incur different delays between consecutive message transmissions [13,17]. Both the Key Hierarchy Problem and the Broadcast Tree problem seek a rooted tree in which the cost for a node may depend on the degrees of the ancestors; however, the optimization objectives are different.

As mentioned in Section 1.1, our approximation algorithm for the general key hierarchy problem uses the elegant algorithm of [14] for finding spanning trees that simultaneously approximates both the minimum spanning tree weight and the shortest path tree weight (from a given root). Such graph structures, commonly referred to as *shallow-light trees* have been extensively studied (e.g., see [1,15]).

2 Problem Definition

An instance of the Key Hierarchy Problem is given by the tuple (S, w, G, c) , where S is the set of group members, $w : S \rightarrow Z$ is the weight function (capturing the update probabilities), $G = (V, E)$ is the underlying communication network with $V \supseteq S \cup \{r\}$ where r is a distinguished node representing the group controller, and $c : E \rightarrow Z$ gives the cost of the edges in G .

Fix an instance (S, w, G, c) . We define a *hierarchy* on a set $X \subseteq S$ to be a rooted tree H whose leaves are the elements of X . For a hierarchy T over X , the cost of a member $x \in X$ with respect to T is given by

$$\sum_{\text{ancestor } u \text{ of } x} \sum_{\text{child } v \text{ of } u} M(T_v) \quad (1)$$

where T_v is the set of leaves in the subtree of T rooted at v and for any set $Y \subseteq S$, $M(Y)$ is the cost of multicasting from the root r to Y in G . The cost of a hierarchy T over X is then simply the sum of the weighted costs of all the members of X with respect to T . The goal of the Key Hierarchy Problem is to determine a hierarchy of minimum cost.

We introduce some notation that is useful for the remainder of the paper. We use $\text{OPT}(S)$ to denote the cost of an optimal hierarchy for S . We extend the notation W to hierarchies and to sets of members: for any hierarchy T (resp., set X of members), $W(T)$ (resp., $W(X)$) denotes the sum of the weights of the leaves of T (resp., members in X). Our algorithms often combine a set $\mathcal{H}$ of two or three hierarchies to another hierarchy T' : $\text{combine}(\mathcal{H})$ introduces a new root

node R , makes the root of each hierarchy in $\mathcal{H}$ as a child of R , and returns the hierarchy rooted at R .

Using the above notation, a more convenient expression for the cost of a hierarchy T over X is the following reorganization of the summation in Equation 1:

$$\sum_{u \in T} W(T_u) = \sum_{\text{child } v \text{ of } u} M(T_v) \quad (2)$$

3 Uniform Multicast Cost

In this section, we consider the special case of the Key Hierarchy problem where the multicast cost to any subset of group members is the same. Thus, the objective is to minimize the average number of multicast messages sent for an update. We note that the number of multicast messages sent for an update at a member u is simply the sum of the degrees of its ancestors in the hierarchy (as is evident from Equation 1).

3.1 Structure of an Optimal Hierarchy for Uniform Member Weights

When all the members have the same weight, we can easily characterize an optimal key hierarchy by recursion. Let n be the number of members. When $n = 1$, the key hierarchy is just a single node tree. When $n = 2$, the key hierarchy is a root with two leaves as children. When $n = 3$, the key hierarchy is a root with three leaves as children. When $n > 3$, we are going to build this key hierarchy recursively. First divide n members into 3 balanced groups, i.e. the size of each group is between $\lfloor n/3 \rfloor$ and $\lceil n/3 \rceil$. Then the key hierarchy is a root with 3 children, each of which is the key hierarchy of one of the 3 groups built recursively by this procedure. It is easy to verify that the cost of this hierarchy is given by:

$$f(n) = \begin{cases} 3n \lfloor \log_3 n \rfloor + 4(n - k) & \text{when } k \leq n < 2k \\ 3n \lfloor \log_3 n \rfloor + 5n - 6k & \text{when } 2k \leq n < 3k \end{cases}$$

The following theorem is due to [9,10], where this scenario arises as a special case of the constrained set selection problem.

Theorem 1 ([9,10]). *For uniform multicast costs and member weights, the above key hierarchy is optimal.*

3.2 A Polynomial-Time Approximation Scheme for Nonuniform Member Weights

We give a polynomial-time approximation scheme for the Key Hierarchy Problem when the multicast cost to every subset of the group is identical and the members have arbitrary weights. Given a positive constant ε , we present an polynomial-time algorithm that produces a $(1 + O(\varepsilon))$ -approximation. We assume that $1/\varepsilon$ is a power of 3; if not, we can replace ε by a smaller constant that satisfies

this condition. We round the weight of every member up to the nearest power of $(1 + \varepsilon)$ at the expense of a factor of $(1 + \varepsilon)$ in approximation. Thus, in the remainder we assume that every weight is a power of $(1 + \varepsilon)$. Our algorithm PTAS(S), which takes as input a set S of members with weights, is as follows.

1. Divide S into two sets, a set H of the $3^{1/\varepsilon^2}$ members with the largest weight and the set $L = S - H$.
2. Initialize $\mathcal{L}$ to be the set of hierarchies consisting of one depth-0 hierarchy for each member of L .
3. Repeat the following step until it can no longer be executed: if T_1, T_2 , and T_3 are hierarchies in $\mathcal{L}$ with identical weight, then replace T_1, T_2 , and T_3 in $\mathcal{L}$ by $\text{combine}(\{T_1, T_2, T_3\})$. (Recall the definition of combine from Section 2.)
4. Repeat the following step until $\mathcal{L}$ has one hierarchy: replace the two hierarchies T_1, T_2 with least weight by $\text{combine}(\{T_1, T_2\})$. Let T_L denote the hierarchy in $\mathcal{L}$.
5. Compute an optimal hierarchy T^* for H . Determine a node in T^* that has weight at most $W(S)\varepsilon$ and height at most $1/\varepsilon$. We note that such a node exists since every hierarchy with at least ℓ leaves has a set N of at least $1/\varepsilon$ nodes at depth at most $1/\varepsilon$ with the property that no node in N is an ancestor of another. Set the root of T_L as the child of this node. Return T^* .

We now analyze the above algorithm. At the end of step 3, the cost of any hierarchy T in $\mathcal{L}$ is equal to $\sum_{v \in T} 3w_v \log_3(W(T)/w_v)$. If $\mathcal{L}$ is the hierarchy set at the end of step 3, then the additional cost incurred in step 4 is at most $\sum_{T \in \mathcal{L}} 2W(T) \log_2(W(L)/W(T))$.

Since there are at most two hierarchies in any weight category in $\mathcal{L}$ at the start of step 4, at least $1 - 1/\varepsilon^2$ of the weight in the hierarchy set is concentrated in the heaviest $4/\varepsilon^3$ hierarchies of $\mathcal{L}$. Step 4 is essentially the Huffman coding algorithm and yields an optimal binary hierarchy. We can show that this binary hierarchy achieves an approximation of 3. This yields the following bound on the increase in cost due to step 4:

$$3(\varepsilon^2 W(L) \log_{1+\varepsilon} 3 + (1 - \varepsilon^2) W(L) \log_2(4/\varepsilon^2)) \leq W(L)/\varepsilon,$$

for ε sufficiently small. The final step of the algorithm increases the cost by at most $W(L)/\varepsilon + \varepsilon W(S)$. Thus, the total cost of the final hierarchy is at most

$$\begin{aligned} & \text{OPT}(H) + \text{OPT}(L) + W(L)/\varepsilon + W(L)/\varepsilon + \varepsilon W(S) \\ & \leq \text{OPT}(H) + \text{OPT}(L) + 2\varepsilon \text{OPT}(S) + \varepsilon \text{OPT}(S) \\ & \leq (1 + 3\varepsilon) \text{OPT}(S). \end{aligned}$$

(The second step holds since $\text{OPT}(S) \geq \sum_{v \in L} w_v \log_3(W(S)/w_v) \geq W(L)/\varepsilon^2$.)

4 Hardness Results

In this section, first we show that Key Hierarchy Problem is strongly NP-complete if group members have nonuniform weights and the underlying routing network is a tree. Then we show the problem is also NP-complete if group members have uniform weights and the underlying routing network is a general graph.

4.1 Weighted Key Hierarchy Problem with Routing Tree

We refer the reader to [5] for the proof of the following theorem.

Theorem 2. *When group members have different weights and the routing network is a tree, the Key Hierarchy Problem is NP-complete.*

4.2 Unweighted Key Hierarchy Problem

We refer the reader to [5] for the proof of the following theorem.

Theorem 3. *When group members have the same key update weights and the routing network is a general graph, the Key Hierarchy Problem is NP-complete.*

5 Approximation Algorithms for Nonuniform Multicast Costs

We first present, in Section 5.1, an 11-approximation algorithm for the case where the underlying communication network is a tree. Then we present, in Section 5.2, a 75-approximation algorithm for the most general case of our problem, where the communication network is an arbitrary weighted graph.

5.1 Approximation Algorithms for Routing Trees

Given any routing tree, let S be the set of members. We start with defining a procedure `partition`(S) that takes as input the set S and returns a pair (X, v) where X is a subset of S and v is a node in the routing tree. First, we determine if there is an internal node v that has a subset C of children such that the total weight of the members in the subtrees of the routing tree rooted at the nodes in C is between $W(S)/3$ and $2W(S)/3$. If v exists, then we partition S into two parts X , which is the set of members in the subtrees rooted at the nodes in C , and $S \setminus X$. It follows that $W(S)/3 \leq W(X) \leq 2W(S)/3$. If v does not exist, then it is easy to see that there is a single member with weight more than $2W(S)/3$. In this case, we set X to be the singleton set that contains this heavy node which we call v . The procedure `partition`(S) returns the pair (X, v) . In the remainder, we let Y denote $S \setminus X$.

ApproxTree(S)

1. If S is a singleton set, then return the trivial hierarchy with a single node.
2. $(X, v) = \text{partition}(S)$; let Y denote $S \setminus X$.
3. Let Δ be the cost from root to partition node v . If $\Delta \leq M(S)/5$, then let $T_1 = \text{ApproxTree}(X)$; otherwise $T_1 = \text{PTAS}(X)$. (PTAS is the algorithm introduced in Section 3.2.)
4. $T_2 = \text{ApproxTree}(Y)$.
5. Return `combine`(T_1, T_2).

Theorem 4. *Algorithm **ApproxTree** is an $(11+\varepsilon)$ -approximation, where $\varepsilon > 0$ can be made arbitrarily small.*

Proof. Let $\text{ALG}(S)$ be the key hierarchy constructed by our algorithm, $\text{OPT}(S)$ be the optimal key hierarchy. In the following proof, we abuse our notation and use $\text{ALG}(\cdot)$ and $\text{OPT}(\cdot)$ to refer to both the key hierarchies and their cost. We notice that $\text{OPT}(S) \geq \text{OPT}(X) + \text{OPT}(Y)$.

We prove by induction on the number of members in S that $\text{ALG}(S) \leq \alpha \cdot \text{OPT}(S) + \beta \cdot W(S)M(S)$, for constants α and β specified later. The induction base case, when $|S| \leq 2$, is trivial. For the induction step, we consider three cases depending on the distance to the partition node v and whether we obtain a balanced partition; we say that a partition (X, Y) is balanced if $\frac{1}{3}W(S) \leq W(X), W(Y) \leq \frac{2}{3}W(S)$. The first case is where $\Delta \leq M(S)/5$ and the partition is balanced. In this case, we have

$$\begin{aligned} \text{ALG}(S) &= \text{ALG}(X) + \text{ALG}(Y) + W(S) [M(X) + M(Y)] \\ &\leq \alpha \cdot \text{OPT}(X) + \beta \cdot W(X)M(X) + \alpha \cdot \text{OPT}(Y) + \beta \cdot W(Y)M(Y) \\ &\quad + W(S) [M(X) + M(Y)] \\ &\leq \alpha [\text{OPT}(X) + \text{OPT}(Y)] + \left(\frac{2}{3}\beta + 1\right) W(S) [M(X) + M(Y)] \\ &\leq \alpha \cdot \text{OPT}(S) + \left(\frac{2}{3}\beta + 1\right) W(S) [M(S) + \Delta] \\ &\leq \alpha \cdot \text{OPT}(S) + \beta \cdot W(S)M(S) \end{aligned}$$

as long as $(1 + \frac{1}{5}) (\frac{2}{3}\beta + 1) \leq \beta$, which is true if $\beta \geq 6$. The second case is where $\Delta > M(S)/5$ and the partition is balanced. In this case, we only call the algorithm recursively on Y and use PTAS on X .

$$\begin{aligned} \text{ALG}(S) &= \text{PTAS}(X) + \text{ALG}(Y) + W(S) [M(X) + M(Y)] \\ &\leq 5(1 + \varepsilon) \cdot \text{OPT}(X) + \alpha \cdot \text{OPT}(Y) + \beta \cdot W(Y)M(Y) \\ &\quad + W(S) [M(X) + M(Y)] \\ &\leq \alpha \cdot \text{OPT}(S) + \left(\frac{2}{3}\beta + 2\right) W(S)M(S) \\ &\leq \alpha \cdot \text{OPT}(S) + \beta \cdot W(S)M(S) \end{aligned}$$

as long as $\alpha \geq 5(1 + \varepsilon)$ and $\frac{2}{3}\beta + 2 \leq \beta$ which is true if $\beta \geq 6$. The third case is when the partition is not balanced (i.e. $W(X) > \frac{2}{3}W(S)$). In this case, our algorithm connects the heavy node directly to the root of the hierarchy.

$$\begin{aligned} \text{ALG}(S) &= \text{ALG}(Y) + W(S) [M(X) + M(Y)] \\ &\leq \alpha \cdot \text{OPT}(Y) + \beta \cdot W(Y)M(Y) + W(S) [M(X) + M(Y)] \\ &\leq \alpha \cdot \text{OPT}(S) + \frac{1}{3}\beta W(S)M(S) + 2W(S)M(S) \\ &\leq \alpha \cdot \text{OPT}(S) + \beta \cdot W(S)M(S) \end{aligned}$$

as long as $\frac{1}{3}\beta + 2 \leq \beta$ which is true if $\beta \geq 3$. So, by induction, we have shown $\text{ALG}(S) \leq \alpha \cdot \text{OPT}(S) + \beta \cdot W(S)M(S)$ for $\alpha \geq 5(1 + \varepsilon)$ and $\beta \geq 6$. Since $\text{OPT}(S) \geq W(S)M(S)$, we obtain an $(11 + \varepsilon)$ -approximation. $\square$

If the member weights are uniform, then we can improve the approximation ratio to 4.2 using a more careful analysis of the same algorithm. We refer the reader to [5] for details.

5.2 Approximation Algorithms for Routing Graphs

In this section, we give a constant-factor approximation algorithm for the case where weights are nonuniform and the routing network is an arbitrary graph. In our algorithm, we compute light approximate shortest-path trees (LAST) [14] of subgraphs of the routing graph. An (α, β) -LAST of a given weighted graph G is a spanning tree T of G such that the shortest path in T from a specified root to any vertex is at most α times the shortest path from the root to the vertex in G , and the total weight of T is at most β times the minimum spanning tree of G .

ApproxGraph(S)

1. If S is a singleton set, return the trivial hierarchy with one node.
2. Compute the complete graph on $S \cup \{root\}$. The weight of an edge (u, v) is the length of shortest path between u and v in the original routing graph.
3. Compute the minimum spanning tree on this complete graph. Call it $MST(S)$.
4. Compute an (α, β) -LAST L of $MST(S)$.
5. $(X, v) = \text{partition}(L)$.
6. Let Δ be the cost from root to partition node L . If $\Delta \leq M(S)/5$, then let $T_1 = \text{ApproxGraph}(X)$. Otherwise, $T_1 = \text{PTAS}(X)$.
7. $T_2 = \text{ApproxGraph}(Y)$.
8. Return $\text{combine}(T_1, T_2)$.

The optimum multicast to a member set is obtained by a minimum Steiner tree, computing which is NP-hard. It is well known that the minimum Steiner tree is 2-approximated by a minimum spanning tree (MST) in the metric space connecting the root to the desired members (the metric being the shortest path cost in the routing graph). So at the cost of a factor 2 in the approximation, we define $M(S)$ to be the cost of the MST connecting the root to S in the complete graph $G(S)$ whose vertex set is $S \cup \{root\}$ and the weight of edge (u, v) is the shortest path distance between u and v in the routing graph.

Theorem 5. *The algorithm **ApproxGraph** is a constant-factor approximation.*

The proof of Theorem 5 is similar to that of Theorem 4. We refer the reader to [5] for the proof details.

References

1. Awerbuch, B., Baratz, A.E., Peleg, D.: Cost-Sensitive Analysis of Communication Protocols. In: PODC (1990)
2. Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., Pinkas, B.: Multicast Security: A Taxonomy and Some Efficient Constructions. In: INFOCOMM (1999)

3. Canetti, R., Malkin, T.G., Nissim, K.: Efficient communication-storage tradeoffs for multicast encryption. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 459–470. Springer, Heidelberg (1999)
4. Caronni, G., Waldvogel, M., Sun, D., Plattner, B.: Efficient Security for Large and Dynamic Multicast Groups. In: WETICE (1998)
5. Chan, A., Rajaraman, R., Sun, Z., Zhu, F.: Approximation Algorithms for Key Management in Secure Multicast, arXiv:0904.4061v1 [cs.DS] (2009)
6. Golin, M.J., Kenyon, C., Young, N.E.: Huffman coding with unequal letter costs. In: STOC (2002)
7. Harney, H., Muckenhirn, C.: Group Key Management Protocol (GKMP) Architecture. Internet RFC 2094 (1997)
8. Harney, H., Muckenhirn, C.: Group Key Management Protocol (GKMP) Specification. Internet RFC 2093 (1997)
9. Heeringa, B.: Improving Access to Organized Information. Thesis, University of Massachusetts, Amherst (2008)
10. Heeringa, B., Adler, M.: Optimal website design with the constrained subtree selection problem. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 757–769. Springer, Heidelberg (2004)
11. Huffman, D.: A Method for the Construction of Minimum-Redundancy Codes. In: IRE (1952)
12. Karp, R.: Minimum-redundancy coding for the discrete noiseless channel. In: IRE Transactions on Information Theory (1961)
13. Khuller, S., Kim, Y.A.: Broadcasting in Heterogeneous Networks. *Algorithmica* 14(1), 1–21 (2007)
14. Khuller, S., Raghavachari, B., Young, N.E.: Balancing Minimum Spanning Trees and Shortest-Path Trees. *Algorithmica* 14(4), 305–321 (1995)
15. Kortsarz, G., Peleg, D.: Approximating Shallow-Light Trees (Extended Abstract). In: SODA (1997)
16. Lazos, L., Poovendran, R.: Cross-layer design for energy-efficient secure multicast communications in ad hoc networks. In: IEEE Int. Conf. Communications (2004)
17. Liu, P.: Broadcast Scheduling Optimization for Heterogeneous Cluster Systems. *J. Algorithms*. 42(1), 135–152 (2002)
18. Mittra, S.: Iolus: A Framework for Scalable Secure Multicasting. In: SIGCOMM (1997)
19. Poovendran, R., Baras, J.S.: An information-theoretic approach for design and analysis of rooted-tree-based multicast key management schemes. In: IEEE Transactions on Information Theory (2001)
20. Salido, J., Lazos, L., Poovendran, R.: Energy and bandwidth-efficient key distribution in wireless ad hoc networks: a cross-layer approach. In: IEEE/ACM Trans. Netw. (2007)
21. Shields, C., Garcia-Luna-Aceves, J.J.: KHIP—a scalable protocol for secure multicast routing. In: SIGCOMM (1999)
22. Snoeyink, J., Suri, S., Varghese, G.: A Lower Bound for Multicast Key Distribution. In: IEEE Infocomm (2001)
23. Wallner, D., Harder, E., Agee, R.: Key Management for Multicast: Issues and Architectures. Internet RFC 2627 (1999)
24. Wong, C.K., Gouda, M.G., Lam, S.S.: Secure Group Communications Using Key Graphs. In: SIGCOMM (1998)

On Smoothed Analysis of Quicksort and Hoare's Find

Mahmoud Fouz¹, Manfred Kufleitner², Bodo Manthey¹,
and Nima Zeini Jahromi¹

¹ Saarland University, Department of Computer Science
Postfach 151150, 66041 Saarbrücken, Germany
{mfouz,manthey}@cs.uni-saarland.de, nzeini@studcs.uni-saarland.de

² Universität Stuttgart, FMI
Universitätsstraße 38, 70569 Stuttgart, Germany
manfred.kufleitner@fmi.uni-stuttgart.de

Abstract. We provide a smoothed analysis of Hoare's find algorithm and we revisit the smoothed analysis of quicksort. Hoare's find algorithm – often called quickselect – is an easy-to-implement algorithm for finding the k -th smallest element of a sequence. While the worst-case number of comparisons that Hoare's find needs is $\Theta(n^2)$, the average-case number is $\Theta(n)$. We analyze what happens between these two extremes by providing a smoothed analysis of the algorithm in terms of two different perturbation models: additive noise and partial permutations.

In the first model, an adversary specifies a sequence of n numbers of $[0, 1]$, and then each number is perturbed by adding a random number drawn from the interval $[0, d]$. We prove that Hoare's find needs $\Theta(\frac{n}{d+1}\sqrt{n/d} + n)$ comparisons in expectation if the adversary may also specify the element that we would like to find. Furthermore, we show that Hoare's find needs fewer comparisons for finding the median.

In the second model, each element is marked with probability p and then a random permutation is applied to the marked elements. We prove that the expected number of comparisons to find the median is in $\Omega((1-p)\frac{n}{p}\log n)$, which is again tight.

Finally, we provide lower bounds for the smoothed number of comparisons of quicksort and Hoare's find for the median-of-three pivot rule, which usually yields faster algorithms than always selecting the first element: The pivot is the median of the first, middle, and last element of the sequence. We show that median-of-three does not yield a significant improvement over the classic rule: the lower bounds for the classic rule carry over to median-of-three.

1 Introduction

To explain the discrepancy between average-case and worst-case behavior of the simplex algorithm, Spielman and Teng introduced the notion of *smoothed analysis* [17]. Smoothed analysis interpolates between average-case and worst-case analysis: Instead of taking a worst-case instance, we analyze the expected worst-case running time subject to slight random perturbations. The more influence

we allow for perturbations, the closer we come to the average case analysis of the algorithm. Therefore, smoothed analysis is a hybrid of worst-case and average-case analysis. In practice, neither can we assume that all instances are equally likely, nor that instances are precisely worst-case instances. The goal of smoothed analysis is to capture the notion of a *typical* instance mathematically. Typical instances are, in contrast to worst-case instances, often subject to measurement or rounding errors. On the other hand, typical instances still have some (adversarial) structure, which instances drawn completely at random do not. Spielman and Teng [18] give a survey of results and open problems in smoothed analysis.

In this paper, we provide a smoothed analysis of Hoare's find [7], which is a simple algorithm for finding the k -th smallest element of a sequence of numbers: Pick the first element as the pivot and compare it to all $n - 1$ remaining elements. Assume that $\ell - 1$ elements are smaller than the pivot. If $\ell = k$, then the pivot is the element that we are looking for. If $\ell > k$, then we recurse to find the k -th smallest element of the list of the smaller elements. If $\ell < k$, then we recurse to find the $(k - \ell)$ -th smallest element among the larger elements. The number of comparisons to find the specified element is $\Theta(n^2)$ in the worst case and $\Theta(n)$ on average. Furthermore, the variance of the number of comparisons is $\Theta(n^2)$ [8]. As our first result, we close the gap between the quadratic worst-case running-time and the expected linear running-time by providing a smoothed analysis.

Hoare's find is closely related to quicksort [6], which needs $\Theta(n^2)$ comparisons in the worst case and $\Theta(n \log n)$ on average [10, Section 5.2.2]. The smoothed number of comparisons that quicksort needs has already been analyzed [12]. Choosing the first element as the pivot element, however, results in poor running-time if the sequence is nearly sorted. There are two common approaches to circumvent this problem: First, one can choose the pivot randomly among the elements. However, randomness is needed to do so, which is sometimes expensive. Second, without any randomness, a common approach to circumvent this problem is to compute the median of the first, middle, and last element of the sequence and then to use this median as the pivot [16, 15]. This method is faster in practice since it yields more balanced partitions and it makes the worst-case behavior much more unlikely [10, Section 5.5]. It is also faster both in average and in worst case, albeit only by constant factors [4, 14]. Quicksort with the median-of-three rule is widely used, for instance in the `qsort()` implementation in the GNU standard C library `glibc` [13] and also in a recent very efficient implementation of quicksort on a GPU [2]. The median-of-three rule has also been used for Hoare's find, and the expected number of comparisons has been analyzed precisely [9]. Our second goal is a smoothed analysis of both quicksort and Hoare's find with the median-of-three rule to get a thorough understanding of this variant of these two algorithms.

1.1 Preliminaries

We denote *sequences* of real numbers by $s = (s_1, \dots, s_n)$, where $s_i \in \mathbb{R}$. For $n \in \mathbb{N}$, we set $[n] = \{1, \dots, n\}$. Let $U = \{i_1, \dots, i_\ell\} \subseteq [n]$ with $i_1 < i_2 < \dots < i_\ell$.

Then $s_U = (s_{i_1}, s_{i_2}, \dots, s_{i_\ell})$ denotes the *subsequence* of s of the elements at positions in U . We denote probabilities by $\mathbb{P}$ and expected values by $\mathbb{E}$.

Throughout the paper, we will assume for the sake of clarity that numbers like $\sqrt{n}$ are integers and we do not write down the tedious floor and ceiling functions that are actually necessary. Since we are interested in asymptotic bounds, this does not affect the validity of the proofs.

Pivot Rules. Given a sequence s , a *pivot rule* simply selects one element of s as the *pivot element*. The pivot element will be the one to which we compare all other elements of s . In this paper, we consider four pivot rules, two of which play only a helper role (the acronyms of the rules are in parentheses):

Classic rule (c): The first element s_1 of s is the pivot element.

Median-of-three rule (m3): The median of the first, middle, and last element is the pivot element, i.e., $\text{median}(s_1, s_{\lceil n/2 \rceil}, s_n)$.

Maximum-of-two rule (max2): The maximum of the first and the last element becomes the pivot element, i.e., $\max(s_1, s_n)$.

Minimum-of-two rule (min2): The minimum of the first and the last element becomes the pivot element, i.e., $\min(s_1, s_n)$.

The first pivot rule is the easiest-to-analyze and easiest-to-implement pivot rule for quicksort and Hoare's find. Its major drawback is that it yields poor running-times of quicksort and Hoare's find for nearly sorted sequences. The advantages of the median-of-three rule has already been discussed above. The last two pivot rules are only used as tools for analyzing the median-of-three rule.

Quicksort, Hoare's Find, Left-to-right Maxima. Let s be a sequence of length n consisting of pairwise distinct numbers. Let p be the pivot element of s according to some rule. For the following definitions, let $L = \{i \in \{1, \dots, n\} \mid s_i < p\}$ be the set of positions of elements smaller than the pivot, and let $R = \{i \in \{1, \dots, n\} \mid s_i > p\}$ be the set of positions of elements greater than the pivot.

Quicksort is the following sorting algorithm: Given s , we construct s_L and s_R by comparing all elements to the pivot p . Then we sort s_L and s_R recursively to obtain s'_L and s'_R , respectively. Finally, we output $s' = (s'_L, p, s'_R)$. The number $\text{sort}(s)$ of comparisons needed to sort s is thus $\text{sort}(s) = (n - 1) + \text{sort}(s_L) + \text{sort}(s_R)$ if s has a length of $n \geq 1$, and $\text{sort}(s) = 0$ when s is the empty sequence. We do not count the number of comparisons needed to find the pivot element. Since this number is $O(1)$ per recursive call for the pivot rules considered here, this does not change the asymptotics.

Hoare's find aims at finding the k -th smallest element of s . Let $\ell = |s_L|$. If $\ell = k - 1$, then p is the k -th smallest element. If $\ell \geq k$, then we search for the k -th smallest element of s_L . If $\ell < k - 1$, then we search for the $(k - \ell)$ -th smallest element of s_R . Let $\text{find}(s, k)$ denote the number of comparisons needed to find the k -th smallest element of s , and let $\text{find}(s) = \max_{k \in [n]} \text{find}(s, k)$.

The number of *scan maxima* of s is the number of maxima seen when scanning s according to some pivot rule: let $\text{scan}(s) = 1 + \text{scan}(s_R)$, and let $\text{scan}(s) = 0$ when s is the empty sequence. If we use the classic pivot rule, the number of

scan maxima is just the number of *left-to-right maxima*, i.e., the number of new maxima that we see if we scan s from left to right. The number of scan maxima is a useful tool for analyzing quicksort and Hoare's find, and has applications, e.g., in motion complexity [3].

We write $\text{c-scan}(s)$, $\text{m3-scan}(s)$, $\text{max2-scan}(s)$, and $\text{min2-scan}(s)$ to denote the number of scan maxima according to the classic, median-of-three, maximum, or minimum pivot rule, respectively. Similar notation is used for quicksort and Hoare's find.

Perturbation Model: Additive noise. The first perturbation model that we consider is *additive noise*. Let $d > 0$. Given a sequence $s \in [0, 1]^n$, i.e., the numbers $s_1, \dots, s_n$ lie in the interval $[0, 1]$, we obtain the perturbed sequence $\bar{s} = (\bar{s}_1, \dots, \bar{s}_n)$ by drawing $\nu_1, \dots, \nu_n$ uniformly and independently from the interval $[0, d]$ and setting $\bar{s}_i = s_i + \nu_i$. Note that $d = d(n)$ may be a function of the number n of elements, although this will not always be mentioned explicitly.

We denote by $\text{scan}_d(s)$, $\text{sort}_d(s)$ and $\text{find}_d(s)$ the (random) number of scan maxima, quicksort comparisons, and comparisons of Hoare's find of $\bar{s}$, preceded by the acronym of the pivot rule used.

Our goal is to prove bounds for the smoothed number of comparisons that Hoare's find needs, i.e., $\max_{s \in [0, 1]^n} \mathbb{E}(\text{c-find}_d(s))$, as well as for Hoare's find and quicksort with the median-of-three pivot rule, i.e., $\max_{s \in [0, 1]^n} \mathbb{E}(\text{m3-find}_d(s))$ and $\max_{s \in [0, 1]^n} \mathbb{E}(\text{m3-sort}_d(s))$. The max reflects that the sequence s is chosen by an adversary.

If $d < 1/n$, the sequence s can be chosen such that the order of the elements is unaffected by the perturbation. Thus, in the following, we assume $d \geq 1/n$. If d is large, the noise will swamp out the original instance, and the order of the elements of $\bar{s}$ will basically depend only on the noise rather than the original instance. For intermediate d , we interpolate between the two extremes.

The choice of the intervals for the adversarial part and the noise is arbitrary. All that matters is the ratio of the sizes of the intervals: For $a < b$, we have $\max_{s \in [a, b]^n} \mathbb{E}(\text{find}_{d \cdot (b-a)}(s)) = \max_{s \in [0, 1]^n} \mathbb{E}(\text{find}_d(s))$. In other words, we can scale (and also shift) the intervals, and the results depend only on the ratio of the interval sizes and the number of elements. The same holds for all other measures that we consider. We will exploit this in the analysis of Hoare's find.

Perturbation Model: Partial Permutations. The second perturbation model that we consider is *partial permutations*, introduced by Banderier, Beier, and Mehlhorn [1]. Here, the elements are left unchanged. Instead, we permute a random subsets of the elements.

Without loss of generality, we can assume that s is a permutation of a set of n numbers, say, $\{1, \dots, n\}$. The perturbation parameter is $p \in [0, 1]$. Any element s_i (or, equivalently, any position i) is marked independently of the others with a probability of p . After that, all marked positions are randomly permuted: Let M be the set of positions that are marked, and let $\pi : M \rightarrow M$ be a permutation drawn uniformly at random. Then $\bar{s}_i = s_{\pi(i)}$ if $i \in M$ and $\bar{s}_i = s_i$ otherwise. If $p = 0$, no element is marked, and we obtain worst-case bounds. If $p = 1$,

all elements are marked, and $\bar{s}$ is a uniformly drawn random permutation. We denote by $\text{pp-find}_p(s)$ the random number of comparisons that Hoare's find needs with the classic pivot rule when s is perturbed.

1.2 Known Results

Additive noise is perhaps the most basic and natural perturbation model for smoothed analysis. In particular, Spielman and Teng added random numbers to the entries of the adversarial matrix in their smoothed analysis of the simplex algorithm [17]. Damerow et al. [3] analyzed the smoothed number of left-to-right maxima of a sequence under additive noise. They obtained upper bounds of $O(\sqrt{\frac{n}{d} \log n} + \log n)$ for a variety of distributions and a lower bound of $\Omega(\sqrt{n} + \log n)$. Manthey and Tantau tightened their bounds for uniform noise to $O(\sqrt{n/d} + \log n)$. Furthermore, they proved that the same bounds hold for the smoothed tree height. Finally, they showed that quicksort needs $O(\frac{n}{d+1} \cdot \sqrt{\frac{n}{d}})$ comparisons in expectation, and this bound is also tight [12].

Banderier et al. [1] introduced partial permutations as a perturbation model for ordering problems like left-to-right maxima or quicksort. They proved that a sequence of n numbers has, after partial permutation, an expected number of $O(\sqrt{\frac{n}{p} \log n})$ left-to-right maxima, and they proved a lower bound of $\Omega(\sqrt{n/p})$ for $p \leq \frac{1}{2}$. This has later been tightened by Manthey and Reischuk [11] to $\Theta((1-p) \cdot \sqrt{n/p})$. They transferred this to the height of binary search trees, for which they obtained the same bounds. Banderier et al. [1] also analyzed quicksort, for which they proved an upper bound of $O(\frac{n}{p} \log n)$.

1.3 New Results

We give a smoothed analysis of Hoare's find under additive noise. We consider both finding an arbitrary element and finding the median. First, we analyze finding arbitrary elements, i.e., the adversary specifies k , and we have to find the k -th smallest element (Section 2). For this variant, we prove tight bounds of $\Theta(\frac{n}{d+1} \sqrt{n/d} + n)$ for the expected number of comparisons. This means that already for very small $d \in \omega(1/n)$, the smoothed number of comparisons is reduced compared to the worst case. If d is a small constant, i.e., the noise is a small percentage of the data values like 1%, then $O(n^{3/2})$ comparisons suffice.

If the adversary is to choose k , our lower bound suggests that we will have either $k = 1$ or $k = n$. The main task of Hoare's find, however, is to find medians. Thus, second, we give a separate analysis of how much comparisons are needed to find the median (Section 2.2). It turns out that under additive noise, finding medians is arguably easier than finding maximums or minimums: For $d \leq 1/2$, we have the same bounds as above. For $d \in (\frac{1}{2}, 2)$, we prove a lower bound of $\Omega(n^{3/2} \cdot (1 - \sqrt{d/2}))$, which again matches the upper bound of Section 2 that of course still applies. For $d > 2$, we prove that a linear number of comparisons suffices, which is considerably less than the $\Omega((n/d)^{3/2})$ general lower bound of Section 2. For the special value $d = 2$, we prove a tight bound of $\Theta(n \log n)$.

Table 1. Overview of bounds for additive noise. The bounds for quicksort and scan maxima with classic pivot rule are by Manthey and Tantau [12]. The upper bounds for Hoare's find in general apply also to Hoare's find for finding the median. Note that, even for large d , the precise bounds for quicksort, Hoare's find, and scan maxima never drop below $\Omega(n \log n)$, $\Omega(n)$, and $\Omega(\log n)$, respectively.

algorithm	$d \leq 1/2$	$d \in (1/2, 2)$	$d = 2$	$d > 2$
quicksort (c)	$\Theta(n\sqrt{n/d})$	$\Theta(n^{3/2})$	$\Theta(n^{3/2})$	$\Theta((n/d)^{3/2})$
quicksort (m3)	$\Omega(n\sqrt{n/d})$	$\Omega(n^{3/2})$	$\Omega(n^{3/2})$	$\Omega((n/d)^{3/2})$
Hoare's find (median, c)	$\Theta(n\sqrt{n/d})$	$\Omega(n^{3/2}(1 - \sqrt{d/2}))$	$\Theta(n \log n)$	$O(\frac{d}{d-2} \cdot n)$
Hoare's find (general, c)	$\Theta(n\sqrt{n/d})$	$\Theta(n^{3/2})$	$\Theta(n^{3/2})$	$\Theta((n/d)^{3/2})$
Hoare's find (general, m3)	$\Theta(n\sqrt{n/d})$	$\Theta(n^{3/2})$	$\Theta(n^{3/2})$	$\Theta((n/d)^{3/2})$
scan maxima (c)	$\Theta(\sqrt{n/d})$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n/d})$
scan maxima (m3)	$\Theta(\sqrt{n/d})$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n})$	$\Theta(\sqrt{n/d})$

Table 2. Overview of bounds for partial permutations. All results are for the classic pivot rule. The results about quicksort, scan maxima, and binary search trees are by Banderier et al. [1] and Manthey and Reischuk [11]. The upper bound for quicksort also holds for Hoare's find, while the lower bound for Hoare's find also applies to quicksort.

algorithm	bound
quicksort	$O((n/p) \log n)$
Hoare's find	$\Omega((1-p)(n/p) \log n)$
scan maxima	$\Theta((1-p)\sqrt{n/p})$
binary search trees	$\Theta((1-p)\sqrt{n/p})$

After that, we aim at analyzing different pivot rules, namely the median-of-three rule. As a tool, we analyze the number of scan maxima under the maximum-of-two, minimum-of-two, and median-of-three rule (Section 3). We essentially show that the same bounds as for the classic rule carry over to these rules. Then we apply these findings to quicksort and Hoare's find (Section 4). Again, we prove a lower bound that matches the lower bound for the classic rule. Thus, the median-of-three does not seem to help much under additive noise. The results concerning additive noise are summarized in Table 1.

Finally, and to contrast our findings for additive noise, we analyze Hoare's find under partial permutations (Section 5). We prove that there exists a sequence on which Hoare's find needs an expected number of $\Omega((1-p) \cdot \frac{n}{p} \cdot \log n)$ comparisons. Since this matches the upper bound for quicksort [1] up to a factor of $O(1-p)$, this lower bound is essentially tight. For completeness, Table 2 gives an overview of the results for partial permutations.

Due to lack of space, proofs are omitted. For complete proofs, we refer to the full version of this paper [5].

2 Smoothed Analysis of Hoare's Find

2.1 General Bounds

In this section, we state tight bounds for the smoothed number of comparisons that Hoare's find needs using the classic pivot rule.

Theorem 1. *For $d \geq 1/n$, we have*

$$\max_{s \in [0,1]^n} \mathbb{E}(\text{c-find}_d(s)) \in \Theta\left(\frac{n}{d+1} \sqrt{n/d} + n\right).$$

Since $\text{find}(s) \leq \text{sort}(s)$ for any s , we already have an upper bound for the smoothed number of comparisons that quicksort needs [12]. This bound is $O\left(\frac{n}{d+1} \cdot \sqrt{n/d} + n \log n\right)$, which matches the bound of Theorem 1 for $d \in O(n^{1/3} \cdot \log^{-2/3} n)$. Thus for the proof of the theorem, $d \in \Omega(n^{1/3} \cdot \log^{-2/3} n)$ remains to be analyzed. The proof of the lower bound is similar to Manthey and Tantau's lower bound proof for quicksort [12].

2.2 Finding the Median

In this section, we provide tight bounds for the special case of finding the median of a sequence using Hoare's find. Somewhat surprisingly, finding the median seems to be easier in the sense that fewer comparisons suffice.

Theorem 2. *Depending on d , we have the following bounds for*

$$\max_{s \in [0,1]^n} \mathbb{E}(\text{c-find}_d(s, \lceil n/2 \rceil)) :$$

For $d \leq \frac{1}{2}$, we have $\Theta(n \cdot \sqrt{n/d})$. For $\frac{1}{2} < d < 2$, we have $\Omega((1 - \sqrt{d/2}) \cdot n^{3/2})$ and $O(n^{3/2})$. For $d = 2$, we have $\Theta(n \cdot \log n)$. For $d > 2$, we have $O(\frac{d}{d-2} \cdot n)$.

The upper bound of $O(n \cdot \sqrt{n/d})$ for $d < 2$ follows from our general upper bound (Theorem 1). For $d \leq \frac{1}{2}$, our lower bound construction for the general bounds also works: The median is among the last $n/2$ elements, which are the big ones. (We might want to have $\lceil n/2 \rceil$ or $n/2 + 1$ large elements to assure this.) The rest of the proof remains the same. For $d > 2$, Theorem 2 states a linear bound, which is asymptotically equal to the average-case bound. Thus, we do not need a lower bound in this case.

First, we state a crucial fact about the value of the median: Intuitively, the median should be around $d/2$ if all elements of s are 0, and it should be around $1 + d/2$ if all elements of s are 1. We make this precise: Independent of the input sequence, the median will be neither much smaller than $d/2$ nor much greater than $1 + d/2$ with high probability.

Lemma 1. *Let $s \in [0,1]^n$, and let $d > 0$. Let $\xi = c\sqrt{\log n/n}$. Let m be the median of $\bar{s}$. Then $\mathbb{P}(m \notin [d/2 - \xi, 1 + d/2 + \xi]) \leq 4 \cdot \exp(-2c^2 \log n/d^2)$.*

The idea to prove the upper bound for $d > 2$ is as follows: Since $d > 2$ and according to Lemma 1 above, it is likely that any element can assume a value greater or smaller than the median. Thus, after we have seen a few number of pivots (for which we “pay” with $O(\frac{d}{d-2}n)$ comparisons), all elements that are not already cut off are within some small interval around the median. These elements are uniformly distributed. Thus, the linear average-case bound applies.

Lemma 2. *Let $d > 2$ be bounded away from 2. Then*

$$\max_{s \in [0,1]^n} \mathbb{E}(\text{c-find}_d(s, \lceil n/2 \rceil)) \in O\left(\frac{d}{d-2} \cdot n\right).$$

3 Scan Maxima with Median-of-Three Rule

The results in this section serve as a basis for the analysis of both quicksort and Hoare's find with the median-of-three rule. In order to analyze the number of scan maxima with the median-of-three rule, we analyze this number with the maximum and minimum of two rules. This is justified since, for every sequence s , we have $\text{max2-scan}(s) \leq \text{m3-scan}(s) \leq \text{min2-scan}(s)$.

The reason for considering max2-scan and min2-scan is that it is hard to keep track where the middle element with median-of-three rule lies: Depending on which element actually becomes the pivot and which elements are greater than the pivot, the new middle position can be on the far left or on the far right of the previous middle. From $\mathbb{E}(\text{max2-scan}_d(s)) \in \Omega(\sqrt{\frac{n}{d}} + \log n)$ and $\mathbb{E}(\text{min2-scan}_d(s)) \in O(\sqrt{\frac{n}{d}} + \log n)$, we get our bounds for m3-scan .

Theorem 3. *For every $d \geq 1/n$, we have*

$$\max_{s \in [0,1]^n} \mathbb{E}(\text{m3-scan}_d(s)) \in \Theta(\sqrt{n/d} + \log n).$$

4 Quicksort and Hoare's Find with Median-of-Three Rule

Now we use our results about scan maxima from the previous section to provide lower bounds for the number of comparisons that quicksort and Hoare's find need using the median-of-three pivot rule. We only give lower bounds here since they match already the upper bounds for the classic pivot rule. We strongly believe that the median-of-three rule does not yield worse bounds than the classic rule and, hence, that our bounds are tight. The goal of this section is to establish a lower bound for Hoare's find, which then carries over to quicksort.

Theorem 4. *For $d \geq 1/n$, we have*

$$\begin{aligned} \max_{s \in [0,1]^n} \mathbb{E}(\text{m3-find}_d(s)) &\in \Omega\left(\frac{n}{d+1} \sqrt{n/d} + n\right) \text{ and} \\ \max_{s \in [0,1]^n} \mathbb{E}(\text{m3-sort}_d(s)) &\in \Theta\left(\frac{n}{d+1} \sqrt{n/d} + n \log n\right). \end{aligned}$$

5 Hoare's Find under Partial Permutations

To complement our findings about Hoare's find, we analyze the number of comparisons subject to partial permutations. For this model, we already have an upper bound of $O(\frac{n}{p} \log n)$, since that bound has been proved for quicksort by Banderier et al. [1]. We show that this is asymptotically tight (up to factors depending only on p) by proving that Hoare's find needs a smoothed number of $\Omega((1-p)\frac{n}{p} \cdot \log n)$ comparisons. The main idea behind the proof of the following theorem is as follows: We aim at finding the median. The first few elements are close to and smaller than the median. Thus, it is unlikely that one of them is permuted further to the left. This implies that all unmarked of the first few elements become pivot elements. Then they have to be compared to many of the $\Omega(n)$ elements larger than the median, which yields our lower bound.

Theorem 5. *Let $p \in (0, 1)$ be a constant. There exist sequences s of length n such that under partial permutations we have*

$$\mathbb{E}(\text{pp-find}_p(s)) \in \Omega((1-p) \cdot \frac{n}{p} \cdot \log n).$$

For completeness, to conclude this section, and as a contrast to Sections 2 and 2.2, let us remark that for partial permutations, finding the maximum using Hoare's find seems to be easier than finding the median: The lower bound constructed above for finding the median requires that there are elements on either side of the element we aim for. If we aim at finding the maximum, all elements are on the same side of the target element. In fact, we believe that for finding the maximum, an expected number of $O(f(p) \cdot n)$ for some function f depending on p suffices.

6 Concluding Remarks

We have shown tight bounds for the smoothed number of comparisons for Hoare's find under additive noise and under partial permutations. Somewhat surprisingly, it turned out that, under additive noise, Hoare's find needs (asymptotically) more comparisons for finding the maximum than for finding the median. Furthermore, we analyzed quicksort and Hoare's find with the median-of-three pivot rule, and we proved that median-of-three does not yield an asymptotically better bound. Let us remark that also the lower bounds for left-to-right maxima as well as for the height of binary search trees [11] can be transferred to median-of-three. The bounds remain equal.

A natural question regarding additive noise is what happens when the noise is drawn according to an arbitrary distribution rather than the uniform distribution. Some first results on this for left-to-right maxima were obtained by Damerow et al. [3]. We conjecture the following: If the adversary is allowed to specify a density function bounded by ϕ , then all upper bounds still hold with $d = 1/\phi$ (the maximum density of the uniform distribution on $[0, d]$ is $1/d$). However, as Manthey and Tantau point out [12], a direct transfer of the results for uniform noise to arbitrary noise might be difficult.

References

1. Banderier, C., Beier, R., Mehlhorn, K.: Smoothed Analysis of Three Combinatorial Problems. In: Rován, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 198–207. Springer, Heidelberg (2003)
2. Cederman, D., Tsigas, P.: A Practical Quicksort Algorithm for Graphics Processors. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 246–258. Springer, Heidelberg (2008)
3. Damerow, V., Meyer auf der Heide, F., Räcke, H., Scheideler, C., Sohler, C.: Smoothed Motion Complexity. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 161–171. Springer, Heidelberg (2003)
4. Erkiö, H.: The worst case permutation for median-of-three quicksort. *The Computer Journal* 27(3), 276–277 (1984)
5. Fouz, M., Kufleitner, M., Manthey, B., Zeini Jahromi, N.: On smoothed analysis of quicksort and Hoare's find. Computing Research Repository, arXiv:0904.3898 [cs.DS] (2009)
6. Hoare, C.A.R.: Algorithm 64: Quicksort. *Comm. ACM* 4(7), 322 (1961)
7. Hoare, C.A.R.: Algorithm 65: Find. *Comm. ACM* 4(7), 321–322 (1961)
8. Kirschenhofer, P., Prodinger, H.: Comparisons in Hoare's find algorithm. *Combin. Probab. Comput.* 7(1), 111–120 (1998)
9. Kirschenhofer, P., Prodinger, H., Martinez, C.: Analysis of Hoare's find algorithm with median-of-three partition. *Random Structures Algorithms* 10(1-2), 143–156 (1997)
10. Knuth, D.E.: *Sorting and Searching*, 2nd edn. The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1998)
11. Manthey, B., Reischuk, R.: Smoothed analysis of binary search trees. *Theoret. Comput. Sci.* 378(3), 292–315 (2007)
12. Manthey, B., Tantau, T.: Smoothed analysis of binary search trees and quicksort under additive noise. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 467–478. Springer, Heidelberg (2008)
13. Schmidt, D.C.: qsort.c. C standard library stdlib within glibc 2.7 (2007), <http://ftp.gnu.org/gnu/glibc/>
14. Sedgewick, R.: The analysis of quicksort programs. *Acta Inform.* 7(4), 327–355 (1977)
15. Sedgewick, R.: Implementing quicksort programs. *Comm. ACM* 21(10), 847–857 (1978)
16. Singleton, R.C.: Algorithm 347: An efficient algorithm for sorting with minimal storage. *Comm. ACM* 12(3), 185–186 (1969)
17. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM* 51(3), 385–463 (2004)
18. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms and heuristics: Progress and open questions. In: *Foundations of Computational Mathematics*, Santander 2005, pp. 274–342. Cambridge University Press, Cambridge (2006)

On an Online Traveling Repairman Problem with Flowtimes: Worst-Case and Average-Case Analysis

Axel Simroth¹ and Alexander Souza²

¹ Fraunhofer Institut IVI Dresden, Germany
axel.simroth@ivi.fraunhofer.de

² Institute for Computer Science, Albert-Ludwigs-Universität Freiburg, Germany
souza@informatik.uni-freiburg.de

Abstract. We consider an online problem where a server operates on an edge-weighted graph G and an adversarial sequence of requests to vertices is released over time. Each request requires one unit of servicetime. The server is free to choose the ordering of service and intends to minimize the total flowtime of the requests.

A natural class of algorithms for this problem are IGNORE algorithms. From worst-case perspective we show that IGNORE algorithms are not competitive for flowtime minimization. From an average-case point of view, we obtain a more detailed picture. In our model, the adversary may still choose the vertices of the requests arbitrarily. But the arrival times are according to a stochastic process (with some rate $\lambda > 0$), chosen by the adversary out of a natural class of processes. The class contains the Poisson-process and (some) deterministic arrivals as special cases. We then show that there is an IGNORE algorithm that is competitive if and only if $\lambda \neq 1$. Specifically, for $\lambda \neq 1$, the expected competitive ratio of the algorithm is within a constant of the length of a shortest cycle that visits all vertices of G . The reason for this is that if $\lambda \neq 1$ the requests either arrive slow enough for our algorithm or too fast even for an offline optimal algorithm. For $\lambda = 1$ the routing-mistakes of the online algorithm accumulate just as in the worst case.

As an additional result, we show how IGNORE tours are constructed optimally in polynomial time, if the underlying graph G is a line.

1 Introduction

In this paper we consider the following online problem, where a server operates on a graph G with non-negative edge-weights. Requests to vertices arrive over time (chosen by an adversary), where each request requires servicetime of one unit. Each request must be served, but the server is free to choose their ordering and intends to minimize the total flowtime. The flowtime of a request is the time between its arrival and departure, i.e., the time spent waiting.

Related Work. This problem is a variant of the online traveling repairman problem which differs from ours in the choice of the objective function: minimization of the total (weighted) *completion* time is required rather than total *flowtime*. The offline version of the completion time problem, where G is a metric and there is a set $R \subseteq V$ of requests known in G , is NP-hard, see Afrati et al. [1]. But if G is a real line, then the problem can be solved optimally in $O(n^2)$ time [1]. The online version of the problem was studied, e.g., by Feuerstein and Stougie [5] and Krumke et al. [7], in the competitive paradigm [10]. There, the objective value achieved by an online algorithm is compared to the optimum value when the whole request sequence is known at the outset. The underlying graph considered in [5] was the real line and a lower bound of $1 + \sqrt{2}$ and a 9-competitive algorithm was shown. These results were improved by [7] giving a lower bound of $7/3$ and a 6-competitive algorithm for general metric spaces.

Models with flowtime are considered, e.g., in the papers of Allulli et al. [2], Krumke et al. [8], Bechetti et al. [3], Hauptmeier et al. [6], and Bonifaci [4]. In [2] the same problem without service times was studied and it was shown that there is no competitive algorithm in that case. If look-ahead is allowed, then there is no competitive algorithm for metric spaces with at least two dimensions [2]. An approach taken in [8] was to restrict the power of the adversary with respect to the admissible vertices, yielding constant competitive algorithms on the real line. The following classical scheduling model was studied in [3]. Jobs with stochastic but unknown service times arrive over time and the objective is to minimize the total flowtime. They proved bounds on the expected value of the competitive ratio of a certain algorithm that depend on parameters of the probability distributions considered. While [8,3,2] are in the competitive paradigm, [6] and [4] are not. In [6] a *reasonable load* model is considered, which roughly means that all requests arriving in a sufficiently large time window can be served in a period of at most the same length. They proved that a natural algorithm for an online dial-a-ride problem called IGNORE can finish within a window of twice this length.

A completely different approach was taken in [4]. The setting here is that a queuing system in which requests arrive and depart is called *stable* if both, the flowtime of each served request and the number of unserved requests at any time is bounded. If the rate of arriving requests is too large, then *no* algorithm yields a stable system. Otherwise, several algorithms, including IGNORE, are stable.

Our Contribution. This paper contributes the following results. First of all, from the worst-case perspective we observe that no deterministic online algorithm can have competitive ratio less than $\text{diam}(G)$, where $\text{diam}(G)$ denotes the diameter of G , i.e., the maximum value of all shortest path lengths between all pairs of vertices. Even worse, but not at all surprising, *no* algorithm of the IGNORE-type is competitive, i.e., has bounded competitive ratio, see Theorem 1.

These negative results for the natural IGNORE-type algorithms from the worst-case perspective justify an average-case model. We assume that the requests arrive according to a stochastic process with a parameter $\lambda > 0$ (and two further natural assumptions), but an adversary remains free to choose the vertices of the

requests. This model captures as special cases (some) deterministic arrivals and Poisson arrivals (that can be observed in many real-world scenarios, e.g., arrival times of phone-calls at a call-center). We give an IGNORE-type algorithm that has expected competitive ratio $c(\lambda) \cdot \text{ham}(G)$ if $\lambda \neq 1$, where $c(\lambda)$ is a constant depending on λ , see Theorem 2. The value of $c(\lambda)$ can be as small as two but diverges as λ tends to one. Thus, for $\lambda \neq 1$, the algorithm *is* competitive. The intuition is as follows: If $\lambda < 1$, then the requests slow enough to be served by our algorithm. If $\lambda > 1$, then they arrive faster than they can be served even by an optimum offline algorithm. For the remaining case $\lambda = 1$, *no* IGNORE-type algorithm is competitive. The reason is that the requests arrive as fast as they can be served by an optimal algorithm, but the online algorithm can be forced to make routing-mistakes that accumulate in the flowtimes of many requests.

As an extension we show that an optimal strategy for each phase of an IGNORE algorithm can be computed in polynomial time if the underlying graph is a line, see Theorem 3.

2 Preliminaries

Throughout the paper, we consider the following model. A server operates on a connected undirected graph $G = (V, E)$ with $n \geq 2$ vertices $V = \{1, \dots, n\}$, edges $e \in E$, and edge-weights $w(e) \geq 0$. There is a distinguished initial vertex, 1 say, where the server is located at time zero. The server moves at unit speed, i.e., the time to pass an edge e is equal to $w(e)$. We define the *distance* w_{ij} between two vertices i, j as the weight of a shortest path connecting i with j . Further, let the *hamiltonicity* $\text{ham}(G)$ be the weight of a shortest, not necessarily simple, circle in G that visits all vertices. We assume that once an algorithm starts traversing an edge, it will continue until the opposite vertex is reached.

At certain points in time $0 \leq a_1 \leq \dots \leq a_m$ requests $R = (r_1, \dots, r_m)$ arrive. Each request has the form $r_j = (a_j, v_j)$, i.e., its *arrival* a_j and a vertex v_j . It is demanded that the server eventually visits the vertex v_j to serve the request, which takes unit time. The time d_j when the request is served is called its *departure*. Hence the request has waited for time $f_j = d_j - a_j$, which is called its *flowtime*.

The server is free to choose the ordering of service and is allowed to wait; these decisions induce a *schedule*. For any schedule S , $F(S) = \sum_{j=1}^m f_j(S)$ denotes its *total flowtime*, where $f_j(S)$ is the flowtime of request r_j induced by S . Our objective function is to minimize the total flowtime. The schedule that minimizes F given the request sequence R in advance is denoted S^* . For ease of notation we usually write F instead of $F(S)$ and F^* instead of $F(S^*)$. An algorithm ALG is called *online* if its decision at any point in time t depends only on the past, i.e., the requests $r_1, \dots, r_j$ with $a_j \leq t$. Otherwise the algorithm is called *offline*. We write $\text{ALG}(R) = F$ and $\text{OPT}(R) = F^*$.

We study two different sources for the request sequence R . Firstly, a malicious deterministic adversary is free to choose R . We measure the quality of schedules produced by an online algorithm ALG facing such an adversary with the

competitive ratio $C^{\text{ALG}} = \sup_R \text{ALG}(R)/\text{OPT}(R)$. If C^{ALG} is not bounded, then the algorithm ALG is *not competitive*.

Secondly, we study the following stochastic adversary. For a fixed $\lambda > 0$, let $D(\lambda)$ be a class of distributions for the arrival times having λ as a parameter. The adversary is free to choose a probability distribution among $D(\lambda)$ and for each arriving request a corresponding vertex (even with knowledge of the future arrivals). We define the *expected competitive ratio* by $E^{\text{ALG}}(\lambda) = \sup_{D(\lambda)} \mathbb{E}[\text{ALG}/\text{OPT}]$, where ALG and OPT denote the respective induced random variables.

Throughout the paper, we consider a family of algorithms, called IGNORE, introduced by Shmoys et al. [9] and previously considered, e.g., in [4,6]. Such an algorithm maintains two memories M and M' that are served in *phases* as follows. Suppose that the algorithm has collected several requests in the memory M . As soon as the algorithm starts serving those, all requests that arrive in the meantime are stored in the memory M' (and are hence ignored in the current phase). After all requests in M have been served, the phase ends and the procedure repeats with changed roles of M and M' . Note that we have not yet specified how the requests in a phase are actually served, which leaves space for various strategies.

3 Worst-Case Analysis

The following negative result – the simple proof is omitted due to space limitations – states that no online algorithm for this problem is competitive within less than $\text{diam}(G)$, where $\text{diam}(G) = \max_{i,j \in V} w_{ij}$ is the *diameter* of G . This bound is not particularly strong, but shows that competitive ratios for the problem must depend on the underlying graph. The situation is worse for IGNORE algorithms: They are not even competitive, see Theorem 1.

Observation 1. *For every online algorithm ALG we have $C^{\text{ALG}} \geq \text{diam}(G)$.*

Theorem 1. *No deterministic IGNORE algorithm is competitive for the online traveling repairman problem with flowtimes.*

Proof. We assume that the IGNORE algorithm starts its first phase immediately when the first request arrives. Furthermore we assume that the algorithm serves the requests in each separate phase optimally. We sketch below how to extend the lower bound to arbitrary IGNORE algorithms.

Consider the graph $G = (\{1, 2\}, \{\{1, 2\}\})$ with $w(\{1, 2\}) = h > 0$, where h is an arbitrary integer. Let the initial vertex of both servers be 1.

We compose a request sequence R out of *blocks* $B_1, B_2, \dots, B_k$ of requests as defined below. The idea is as follows: The *delay* of the algorithm at a block B_i is the difference between the respective minimum times the algorithm and the optimum start serving any of the requests in B_i . The essential property of the construction is that the algorithm is forced to traverse the edge twice as often as the optimum. This increases its delay and yields unbounded competitive ratio.

We encode the requests with the format (δ, v) , where v denotes the respective vertex and δ the difference between the arrival times of a request and its predecessor (respectively time zero if there is no predecessor).

$$\begin{aligned}
B_1 &= (h, 2) \\
B_2 &= \underbrace{(1, 2), \dots, (1, 2)}_{h \text{ requests at } 2}, \quad \underbrace{(1, 1)}_{\text{one trap request at } 1}, \quad \underbrace{(1, 2), \dots, (1, 2)}_{3h+1 \text{ requests at } 2} \\
B_3 &= \underbrace{(h+2, 1), (1, 1), \dots, (1, 1)}_{5h-1 \text{ requests at } 1}, \quad \underbrace{(1, 2)}_{\text{one trap request at } 2}, \quad \underbrace{(1, 1), \dots, (1, 1)}_{5h+1 \text{ requests at } 1} \\
B_4 &= \underbrace{(h+2, 2), (1, 2), \dots, (1, 2)}_{9h-1 \text{ requests at } 2}, \quad \underbrace{(1, 1)}_{\text{one trap request at } 1}, \quad \underbrace{(1, 2), \dots, (1, 2)}_{7h+1 \text{ requests at } 2} \\
&\vdots
\end{aligned}$$

Now we show that the delay increases by $2h$ with every block.

The optimum serves the sequence $R = (B_1, B_2, \dots, B_k)$ as follows. At time zero its server travels to vertex 2, arrives there at time h , i.e., just in time to serve the first request with flowtime 1 at time $h+1$. Now the server is at the right vertex to serve all the requests at this vertex of the first block, except for the trap-request. This is the last request of the block to be served after traveling to vertex 1. Then the server is at the right spot to handle the requests of the next block. We continue in this manner for every block. All requests, except traps, are served at flowtime 1, each. The trap at block B_i , say, has flowtime $(2i-1)h + h + 1$. This yields total flowtime

$$\text{OPT}(R) \leq 1 + \sum_{i=2}^k ((4(i-2) + 1)h + (2i-1)h + 2ih + 2) \leq 8hk^2.$$

Now we consider the algorithms. Its server is at vertex 1 until the first request arrives which is served with flowtime $h+1$ at time $2h+1$. Notice that the delay of the algorithm is already h . Until time $2h+1$ the first $h+1$ requests of B_2 have arrived. These include the trap request at vertex 1. As we are dealing with an IGNORE algorithm it will serve the trap during the next phase. As the phases are served optimally the trap is handled last in the phase after one edge-traversal. But now the server is at the wrong vertex to serve the remaining requests of the block and the edge needs to be traversed once more. Thus the algorithm crossed the edge two times more than the optimum server and hence its delay increased by $2h$. We continue in this manner and see that the total delay before block B_i is (at least) $2h(i-1)$. Thus we find the lower bounds on the total flowtime

$$\text{ALG}(R) \geq \sum_{i=1}^k 2h(i-1)(2ih) \geq \sum_{i=k/2}^k 2h^2 i^2 \geq h^2 k^3 / 2$$

and the competitive ratio $C^{\text{ALG}} \geq kh/16$, which can be made arbitrarily large.

To adapt the lower bound for general IGNORE algorithms first recall that any such algorithm waits for a determined time (which may depend on the requests seen so far) before starting the first phase. In this case we start by giving the block B_1 , i.e., one request at vertex 2 at time h which is served by the optimal server at time $h+1$. Then we wait for the server of the algorithm to move. Right at this time we issue the blocks $B_2, B_3, \dots, B_k$ similarly as before.

We can also remove the assumption that the IGNORE algorithm serves each phase optimally. If this is not the case we adapt the blocks as follows, where we use that the algorithm is deterministic: Let t_1 be the time the algorithm needs to finish block B_1 . Similarly to B_2 we issue requests to vertex 2 and one trap requests to vertex 1 in the time until t_1 . If the delay of the algorithm after this phase is less than $2h$ we continue similarly to the remainder of block B_2 , otherwise we continue with block B_3 and so forth. $\square$

4 Average-Case Analysis

In this section we consider the following stochastic variant of the problem. As before, each request requires time one for service. An adversary has to specify a stochastic process $(X_t)_{t \geq 0}$ (satisfying the conditions given below), where X_t denotes the number of arrivals up to some time t . Upon arrival of a new request, the adversary is free to choose the vertex for it, where knowledge of the future arrivals may be used. Recall that a random variable T is called *stopping time* for $(X_t)_{t \geq 0}$ if the event $T = t$ depends only on $(X_s)_{s \leq t}$. The adversarial process has to satisfy the following conditions, where $\lambda > 0$ is a known parameter:

- (1) For $\lambda \leq 1$ and two stopping times $S \leq T$ we have $\mathbb{E}[X_T - X_S] \leq \lambda \mathbb{E}[T - S]$.
- (2) For $\lambda > 1$ and two stopping times $S \leq T$ we have $\mathbb{E}[X_T - X_S] \geq \lambda \mathbb{E}[T - S]$.
- (3) For two stopping times $S \leq T$ we have $\text{Var}[X_T - X_S] \leq c \mathbb{E}[X_T - X_S]$ for some constant c , i.e., the process has to have bounded variance.

Notice that all processes with independent, identically distributed inter-arrival times with expectation λ (and bounded variance) are members of this class. Especially, exponential (Poisson process) and uniform distributed inter-arrival times are covered. Furthermore, if the X_t are deterministic, define $\lambda = \lim_{t \rightarrow \infty} X_t/t$. If this limit exists, also deterministic sequences $(X_t)_{t \geq 0}$ that satisfy the above are admissible.

We consider the expected competitive ratio $E^{\text{ALG}}(\lambda) = \lim_{t \rightarrow \infty} E_t^{\text{ALG}}(\lambda)$, where $E_t^{\text{ALG}}(\lambda) = \mathbb{E}[\text{ALG}/\text{OPT}]$ when the process is stopped at some time t and the expectation is taken with respect to X_t . We show that, in this model, bounded expected competitive ratios are possible for IGNORE algorithms if and only if $\lambda \neq 1$. For the case $\lambda = 1$ the ratio is unbounded for all deterministic IGNORE algorithms.

In specific, we consider the following IGNORE algorithm ALG. At time zero, the algorithm computes a *fixed* circle in the graph G that visits all vertices and has length equal to $\text{ham}(G)$. Let w be a parameter to be specified later on. The algorithm waits for time w and then serves the requests that arrived using

the precomputed tour. All requests that arrive in the meantime are ignored and served in the subsequent phase, etc.

Theorem 2. *For any stochastic process as defined above, if $\lambda \neq 1$, then there is a constant $c(\lambda)$ and an algorithm ALG such that $E^{\text{ALG}}(\lambda) \leq c(\lambda) \cdot \text{ham}(G)$. There is a process with $\lambda = 1$ such that no IGNORE algorithm is expected competitive.*

The statement for $\lambda = 1$ of the theorem follows from the construction in the proof of Theorem 1 noting that $\lambda = \lim_{t \rightarrow \infty} X_t/t = 1$: Then (1) is satisfied since $\mathbb{E}[X_t - X_s] = X_t - X_s \leq t - s$ hold for any two times $s \leq t$. Property (3) is also true since $\text{Var}[X_t - X_s] = 0$ for any times $s \leq t$. The statement for $\lambda \neq 1$ follows directly from Lemma 2 and Lemma 3. The ideas behind these are as follows: For $\lambda < 1$ the requests arrive slow enough to be served within the claimed time-bounds, i.e., the algorithm is competitive. For $\lambda > 1$ they arrive faster than they can be served, i.e., even the optimum is large.

Now we state the following technical lemma – the main tool for our analysis – which relates $\mathbb{E}[\text{ALG}/\text{OPT}]$ with $\mathbb{E}[\text{ALG}]/\mathbb{E}[\text{OPT}]$.

Lemma 1. *Let X and Y be two positive random variables. Then for every $\delta > 0$ such that $\Pr[Y \geq \delta \mathbb{E}[Y]] > 0$ we have*

$$\mathbb{E}\left[\frac{X}{Y}\right] \leq \frac{\mathbb{E}[X]}{\delta \mathbb{E}[Y]} + \mathbb{E}\left[\frac{X}{Y} \mid Y < \delta \mathbb{E}[Y]\right] \Pr[Y < \delta \mathbb{E}[Y]].$$

Below we use the following additional notation. A request r_j is called *pending* while it has arrived but not departed. By definition, the algorithm initially waits until time $T_0 = w$. Let the number of requests that arrive during the time interval $I_0 = [0, T_0)$ be A_0 . The time when the algorithm has served these requests and is back at the initial vertex 1 be T_1 . Further let the number of arriving requests in the time interval $I_1 = [T_0, T_1)$ be A_1 . This induces a sequence of intervals $I_0, I_1, I_2, \dots$, called *phases*, *stopping times* $T_0, T_1, T_2, \dots$ and numbers $A_0, A_1, A_2, \dots$, called *arrivals*. F_j denotes the (random) flowtime of a request r_j and $I(j)$ the phase in which r_j arrives. The optimum flowtime of request r_j is denoted F_j^* .

Lemma 2. *For $\lambda < 1$ we have*

$$E^{\text{ALG}}(\lambda) \leq 3 \left(\frac{1}{1-\lambda} + 1 + 2c \right) \cdot (\text{ham}(G) + 1)$$

Proof. We begin by choosing a value for the parameter w , i.e., the time the algorithm waits before starting its first phase. For any positive value of w , we expect that at most λw requests arrive while waiting. Hence we expect that the first phase takes time at most $\text{ham}(G) + \lambda w$. We require that this time be no more than w , i.e., we need $\text{ham}(G) + \lambda w \leq w$. Hence we choose $w = \text{ham}(G)/(1 - \lambda)$ which is possible for $\lambda < 1$.

Fix an integer k and consider phases $I_0, \dots, I_k$ of the algorithm and the arrivals $A_0, \dots, A_k$. Also introduce the notation $S_k = \sum_{i=0}^k A_i$.

The optimum flowtime for any request r_j that has arrived is $F_j^* \geq 1$. This implies $F^* \geq S_k$. For F_j we have $F_j \leq w + \text{ham}(G) + A_0 + 1$ if $I(j) = 0$ and $F_j \leq \text{ham}(G) + A_{I(j)-1} + \text{ham}(G) + A_{I(j)} + 1$ otherwise. Observe that for any $i \geq 1$ we have $A_{i-1}A_i \leq \max\{A_{i-1}, A_i\}^2 \leq A_{i-1}^2 + A_i^2$. This yields

$$F \leq A_0(w + \text{ham}(G) + 1) + \sum_{i=1}^k (2\text{ham}(G) + 1)A_i + 3 \sum_{i=1}^k A_i^2$$

and hence $\mathbb{E}[F] \leq (w + \text{ham}(G) + 1)\mathbb{E}[A_0] + (2\text{ham}(G) + 1)\sum_{i=1}^k \mathbb{E}[A_i] + 3\sum_{i=1}^k (\text{Var}[A_i] + \mathbb{E}[A_i]^2)$.

We prove by induction that $\mathbb{E}[A_i] \leq \lambda w$ for all $i \geq 0$. The base case $\mathbb{E}[A_0] \leq \lambda w$ is clear. For the inductive case we exploit the properties of our process. By induction hypothesis $\mathbb{E}[A_i] \leq \lambda w$ we have $\mathbb{E}[A_{i+1}] = \mathbb{E}[X_{T_i} - X_{T_{i-1}}] \leq \lambda \mathbb{E}[T_{i+1} - T_i] \leq \lambda \mathbb{E}[\text{ham}(G) + A_i] \leq \lambda(\text{ham}(G) + \lambda w) \leq \lambda w$.

Using $\text{Var}[A_i] = \text{Var}[X_{T_i} - X_{T_{i-1}}] \leq c\mathbb{E}[X_{T_i} - X_{T_{i-1}}] = c\mathbb{E}[A_i]$ we find

$$\begin{aligned} \mathbb{E}[F] &\leq (w + \text{ham}(G) + 1)\mathbb{E}[A_0] + (2\text{ham}(G) + 1)\sum_{i=1}^k \mathbb{E}[A_i] \\ &\quad + 3\sum_{i=1}^k (c + \lambda w)\mathbb{E}[A_i] \\ &\leq 3\left(\frac{1}{1-\lambda} + 1 + c\right)(\text{ham}(G) + 1)\mathbb{E}[S_k]. \end{aligned}$$

The crude estimate $F \leq S_k^2(\text{ham}(G) + 1)$ gives

$$\mathbb{E}\left[\frac{F}{S_k} \mid S_k < \frac{1}{2}\mathbb{E}[S_k]\right] \leq \frac{\mathbb{E}[S_k](\text{ham}(G) + 1)}{2}.$$

With Lemma 1, Chebyshev's inequality, and the choice $\delta = 1/2$ we have

$$\begin{aligned} \mathbb{E}\left[\frac{F}{F^*}\right] &\leq \mathbb{E}\left[\frac{F}{S_k}\right] \leq \frac{2\mathbb{E}[F]}{\mathbb{E}[S_k]} + \mathbb{E}\left[\frac{F}{S_k} \mid S_k < \frac{1}{2}\mathbb{E}[S_k]\right] \Pr\left[S_k < \frac{1}{2}\mathbb{E}[S_k]\right] \\ &\leq \frac{2\mathbb{E}[F]}{\mathbb{E}[S_k]} + \frac{\mathbb{E}[S_k](\text{ham}(G) + 1)}{2} \Pr\left[|S_k - \mathbb{E}[S_k]| \geq \frac{1}{2}\mathbb{E}[S_k]\right] \\ &\leq 3\left(\frac{1}{1-\lambda} + 1 + c\right)(\text{ham}(G) + 1) + 2\mathbb{E}[S_k](\text{ham}(G) + 1) \cdot \frac{\text{Var}[S_k]}{\mathbb{E}[S_k]^2} \end{aligned}$$

yielding the claim using the assumption $\text{Var}[S_k] \leq c\mathbb{E}[S_k]$. $\square$

Lemma 3. *Let $\lambda > 1$ and $0 < \delta < 1$ be an any constant such that $(1 - \delta)\lambda > 1$. Then we have*

$$E^{\text{ALG}}(\lambda) \leq \left(\frac{2}{\delta^2} + \frac{c\lambda(1 - \delta)}{(\lambda(1 - \delta) - 1)^2}\right) \cdot (\text{ham}(G) + 1).$$

Proof. Here we choose $w = 0$, i.e., the algorithms begins its first phase immediately after the arrival of the first request. We stop the process at any fixed time t and let X_t be the number of requests that arrived.

As upper bound we use the crude estimate $F \leq X_t^2(\text{ham}(G) + 1)$. Now we establish the lower bound $F^* \geq X_t^2\delta^2/2$ which holds conditional on the event that $X_t(1 - \delta) \geq \mathbb{E}[X_t]/\lambda$. Observe that the properties of the process and the condition imply $X_t(1 - \delta) \geq t$ and hence $X_t - t \geq \delta X_t$. At time t the optimum can have served at most t many requests. Thus, by the condition on X_t , there are $P \geq X_t - t \geq \delta X_t$ requests pending. As each request takes time at least one we have $F^* \geq \sum_{i=1}^P i \geq \delta^2 X_t^2/2$. Thus, using Chebyshev's inequality, and $\text{Var}[X_t] \leq c\mathbb{E}[X_t]$ we find

$$\Pr[X_t(1 - \delta) < \mathbb{E}[X_t]/\lambda] \leq \frac{c}{\mathbb{E}[X_t] \left(1 - \frac{1}{\lambda(1-\delta)}\right)^2}$$

and hence

$$\begin{aligned} E_t^{\text{ALG}}(\lambda) &\leq \mathbb{E} \left[\frac{2X_t^2(\text{ham}(G) + 1)}{\delta^2 X_t^2} \mid X_t(1 - \delta) \geq \mathbb{E}[X_t]/\lambda \right] \\ &\quad + \mathbb{E} \left[\frac{X_t^2(\text{ham}(G) + 1)}{X_t} \mid X_t(1 - \delta) < \mathbb{E}[X_t]/\lambda \right] \\ &\quad \cdot \Pr[X_t(1 - \delta) < \mathbb{E}[X_t]/\lambda] \\ &\leq \frac{2(\text{ham}(G) + 1)}{\delta^2} + \frac{(\text{ham}(G) + 1)\mathbb{E}[X_t]}{\lambda(1 - \delta)} \cdot \frac{c}{\mathbb{E}[X_t] \left(1 - \frac{1}{\lambda(1-\delta)}\right)^2}, \end{aligned}$$

yielding the claim. $\square$

5 An Optimal Phase-Strategy for the Line

In this section we show that the optimal schedule of an IGNORE-phase can be computed in polynomial time for the special case when the underlying graph G is a line. A *line* is a graph $G = (V, E)$ with vertices $V = \{1, 2, \dots, n\}$ and edges $E = \{\{1, 2\}, \{2, 3\}, \dots, \{n-1, n\}\}$ with weights $w(e) \geq 0$.

Theorem 3. *An optimal schedule of an IGNORE-phase on a line G with n vertices can be computed in $O(n^2)$ time.*

Proof. We give a reduction to the offline traveling repairman problem with total completion time objective, as studied by Afrati et al. [1]. For lines, they proved that this problem can be solved in polynomial time. However, in their setting, the number of requests is bounded by the number of vertices, which does not in general hold in our model.

We may assume that for any vertex i , there are initially $\varrho(i) > 0$ requests pending. Because all requests in the memory M are known at the beginning of each phase, computing the minimum total flow time $F(M) = \sum_{r_j \in M} f_j$ is

equivalent to finding a schedule with minimum total completion time $C(M) = \sum_{r_j \in M} d_j$ (as the arrival times a_j are fixed). The following claim, which can be proved with an exchange argument, reduces the number of requests we have to consider.

Claim. There is an optimal schedule serving all $\varrho(i)$ requests at its first visit to vertex i .

With the above claim, for finding an optimal schedule we can replace all requests at any vertex with a single request (with respective service time). Then the problem is reduced to the traveling repairman problem with completion time objective and we can adapt the $O(n^2)$ dynamic programming approach given in [1]. $\square$

References

1. Afrati, F., Cosmadakis, S., Papadimitriou, C., Papageorgiou, G., Papakonstantinou, N.: The complexity of the travelling repairman problem. *Informatique Theorique et Applications* 20(1), 79–87 (1986)
2. Allulli, L., Ausiello, G., Bonifaci, V., Laura, L.: On the power of lookahead in on-line server routing problems. *Theoretical Computer Science* 408(2-3), 116–128 (2008)
3. Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A., Schäfer, G., Vredeveld, T.: Average case and smoothed competitive analysis of the multilevel feedback algorithm. *Mathematics of Operations Research* 31(1), 85–108 (2006)
4. Bonifaci, V.: An adversarial queueing model for online server routing. *Theoretical Computer Science* 381(1-3), 280–287 (2007)
5. Feuerstein, E., Stougie, L.: On-line single server dial-a-ride problems. *Theoretical Computer Science* 268(1), 91–105 (2001); *Online Algorithms 1998*, Udine, Italy (September 1998)
6. Hauptmeier, D., Krumke, S.O., Rambau, J.: The online dial-a-ride problem under reasonable load. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) *CIAC 2000*. LNCS, vol. 1767, pp. 125–136. Springer, Heidelberg (2000)
7. Krumke, S.O., de Paepe, W.E., Poensgen, D., Stougie, L.: News from the online traveling repairman. *Theoretical Computer Science* 295(1-3), 279–294 (2003)
8. Krumke, S.O., Laura, L., Lipmann, M., Marchetti-Spaccamela, A., de Paepe, W.E., Poensgen, D., Stougie, L.: Non-abusiveness Helps: An $\mathcal{O}(1)$ -Competitive Algorithm for Minimizing the Maximum Flow Time in the Online Traveling Salesman Problem. In: Jansen, K., Leonardi, S., Vazirani, V.V. (eds.) *APPROX 2002*. LNCS, vol. 2462, pp. 200–214. Springer, Heidelberg (2002)
9. Shmoys, D.B., Wein, J., Williamson, D.P.: Scheduling Parallel Machines On-line. *SIAM Journal on Computing* 24(6), 1313–1331 (1995)
10. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. *Communications of the ACM* 28(2), 202–208 (1985)

Three New Algorithms for Regular Language Enumeration

Margareta Ackerman¹ and Erkki Mäkinen²

¹ University of Waterloo, Waterloo ON, Canada

² University of Tampere, Tampere, Finland
em@cs.uta.fi

Abstract. We present new and more efficient algorithms for regular language enumeration problems. The *min-word problem* is to find the lexicographically minimal word of length n accepted by a given NFA, the *cross-section problem* is to list all words of length n accepted by an NFA in lexicographical order, and the *enumeration problem* is to list the first m words accepted by an NFA according to length-lexicographic order. For the min-word and cross-section problems, we present algorithms with better asymptotic running times than previously known algorithms. Additionally, for each problem, we present algorithms with better practical running times than previously known algorithms.

1 Introduction

We would like to explore the language accepted by a given NFA by obtaining a sample of words from that language. The *min-word problem* is to find the lexicographically minimal word of length n accepted by a given NFA. The *cross-section problem* is to list all words of length n accepted by an NFA in lexicographical order. The *enumeration problem* is to list the first m words accepted by an NFA according to length-lexicographic order¹ (sorting a set of words according to length-lexicographic order is equivalent to sorting words of equal length according to lexicographic order and then sorting the set by length using a stable sort).

We can use algorithms for the above problems to test the correctness of NFAs and regular expressions. (If a regular language is represented via a regular expression, we first convert it to an NFA.) While such a technique provides evidence that the correct NFA or regular expression has been found, an algorithm for the enumeration problem can also be used to fully verify correctness once sufficiently many words have been enumerated [3, p.11].

An algorithm for the cross-section problem can be used to decide whether every word accepted by a given NFA on s states is a power (a string of the form x^ℓ , for $|x| \geq 1$ and $\ell \geq 2$). It was shown by Anderson et. al. [2] that if every word accepted by the NFA is a power, then the NFA accepts no more than $7s$

¹ Length-lexicographic order is also known as “radix order” and “pseudo-lexicographic order.”

words of each length, and further, if it accepts a non-power, it must accept a non-power of length less than $3s$. Using these results, Anderson et. al. [2] get an efficient algorithm for determining whether every word accepted by an NFA is a power by enumerating all the words of length $1, 2, \dots, 3s - 1$ and testing if each word is a power, stopping if the length of any cross-section exceeds $7s$.

The cross-section problem also leads to an alternative solution to the next k -subset of an n -set problem. The problem is, given a set $T = \{e_1, e_2, \dots, e_n\}$, to enumerate all k -subsets of T in alphabetical order. See [1] for details of the solution.

For the min-word and cross-section problems, we present algorithms with better asymptotic and practical running times than previously known algorithms. In addition, for the enumeration problem, we present an algorithm with better practical running time than previously known algorithms, and the same asymptotic running time as the optimal previously known algorithm.

We analyze the algorithms in terms of their output size, the parameters of the NFA, and the length of words in the cross-section for the cross-section enumeration algorithms. The output size, t , is the total number of characters over all words enumerated by the algorithm. An NFA is a five-tuple $N = (Q, \Sigma, \delta, q_0, F)$ where Q is the set of states, Σ is the alphabet, δ is the transition function, q_0 is the start state, and F is the set of final states. In our analysis we consider $s = |Q|$, $\sigma = |\Sigma|$, and d , the number of transitions in the NFA. We assume that the graph induced by the NFA is connected (otherwise, we can preprocess the NFA in $O(s + d)$ operations.) Our algorithms are more efficient than previous algorithms in terms of the number of states and transitions of the NFA, without compromising efficiency on the other parameters.

First we present some previous work on regular language enumeration, as well as set up the framework which we will use for most of our new algorithms. Next, we present our first set of algorithms, which we refer to as the AMSorted enumeration algorithms, with better asymptotic running times than previous algorithms for the min-word and cross-section problems. Next, we present the AMBoolean enumeration algorithms, with even better asymptotic running times for the min-word and cross-section problems. Lastly, we present a very simple set of algorithms, which we call IntersectionEnumeration, which gives a min-word and a cross-section algorithm with the same asymptotic running time as the AMBoolean algorithms (however, the IntersectionEnumeration algorithms are inefficient in practice). We perform rigorous testing of our algorithms and previous enumeration algorithms. We find that the AMSorted algorithm for min-word and the AMBoolean algorithm for the cross-section and enumeration problems have the best practical running times.

2 Previous Work

The lookahead-matrix cross-section enumeration algorithm presented by Ackerman and Shallit [1] is the most efficient previously known algorithm in terms of n , the length of the words in the cross-section. As shown in [1], the cross-section lookahead-matrix algorithm, `crossSectionLM` is $O(s^{2.376}n + \sigma s^2 t)$ [1]. The

min-word lookahead-matrix algorithm, **minWordLM**, finds the minimal word of length n in $O(s^{2.376}n)$ time and $O(s^2n)$ space [1]. The algorithm **enumLM**, the lookahead-matrix algorithm for the enumeration problem, uses $O(s^{2.376}c + \sigma s^2t)$ operations, where c is the number of cross-sections encountered throughout the enumeration.

We now analyze these algorithms with respect to d , the number of edges in the NFA. The lookahead-matrix algorithms use the framework in Section 3. As described in Section 3, after the minimal word in the cross-section has been found, to enumerate a cross-section we need to examine the transitions from all the states currently on the state stack (of which there are at most d). Thus, **enumCrossSection** is $O(s^{2.376}n + dt)$. Similarly, **enumLM** is $O(s^{2.376}c + dt)$.

Another previous set of algorithms is Mäkinen's algorithms, originally presented in [5] and analyzed in the unit-cost model, where it is linear in n . In the bit-complexity model, Mäkinen's cross-section enumeration algorithm is quadratic in n . In [1], Ackerman and Shallit discuss the theoretical and practical performance of two versions of Mäkinen's algorithms, MäkinenI (MI) and MäkinenII (MII). The main difference between these two algorithms is the way how they determine when a given cross-section has been fully enumerated. The MI cross-section algorithm precomputes the maximal word in a given cross-section, and terminates when the maximal word is found in the enumeration. The MII cross-section algorithm terminates when the state stack is empty (the same termination method is used in **cross-section** in Section 3). While the two versions of Mäkinen's algorithm have the same asymptotic running time, the MII cross-section algorithm has better practical performance. The asymptotic running time of Mäkinen's min-word algorithm is $\Theta(s^2n^2)$, as shown in [1]. Mäkinen's cross-section algorithms are $O(s^2n^2 + \sigma s^2t)$, and Mäkinen's enumeration algorithms are $O(\sigma s^2t + s^2e)$, where e is the number of empty cross-sections encountered throughout the enumeration [1]. If we analyze these algorithms with respect to d , the number of edges in the NFA, we find that Mäkinen's min-word algorithms are $\Theta(dn^2)$, the cross-section algorithms are $O(d(n^2 + t))$ and the enumeration algorithms are $O(d(e + t))$. When referring to Mäkinen's algorithm, we use the MII versions, calling them **minWordM**, **crossSectionM**, and **enumM**.

The Grail computation package [4] includes a cross-section enumeration algorithm, **fmenum**, that uses a breadth-first-search approach on the tree of all possible paths from the start state of the NFA [4]. The algorithm is exponential in n [1].

It was found that **crossSectionM** and **enumM** usually have the best running time in practice. In special cases, where the quadratic running time of Mäkinen's cross-section algorithm is reached, lookahead-matrix performs better [1].

For all algorithms, we assume that the characters on the transitions from state A to state B are sorted, for all states A and B . Then the running time is independent of alphabet size. Otherwise, we can sort the characters on transitions between all pairs of states in $O(\sigma \log \sigma s^2)$ operations.

3 Algorithm Framework

We present the general framework commonly used for cross-section and enumeration algorithms. This framework was used for the lookahead-matrix algorithms and Mäkinen algorithms [1]. We use this framework for most of our new algorithms.

For cross-section enumeration, we first find the minimal word $w = a_1a_2 \cdots a_n$ in the cross-section with respect to length-lexicographic order, or determine that the cross-section is empty.

Definition 1 (*i*-complete). *We say that state q of NFA N is i -complete if starting from q there is a path in N of length exactly i ending at a final state of N .*

Let $S_0 = \{q_0\}$ and $S_i = \cup_{q \in S_{i-1}} \delta(q, a_i) \cap \{q \mid q \text{ is } (n-i)\text{-complete}\}$, for $1 \leq i < n$. That is, S_i is the set of $(n-i)$ -complete states reachable from the states in S_{i-1} on a_i . We find w while storing the sets of states $S_0, S_1, S_2, \dots, S_{n-1}$ on the *state stack*, S , which we assume is global.

We assume that there is some implementation of the method $\text{minWord}(n, N)$, which returns the minimal word w of length n accepted by NFA N starting from one of the states on top of the state stack, or returns NULL if no such word exists. Different algorithms use different approaches for finding minimal words. To find the next word, we scan the minimal word $a_1a_2 \cdots a_n$ from right to left, looking for the shortest suffix that can be replaced such that the new word is in $L(N)$. It follows that the suffix $a_i \cdots a_n$ can be replaced if the set of $(n-i)$ -complete states reachable from S_{i-1} on any symbol greater than a_i is not empty. As we search for the next word of length n , we update the state stack. Therefore, each consecutive word can be found using the described procedure. The algorithm is outlined in detail in `nextWord`. Note that the algorithms use indentation to denote the scope of loops and if-statements.

Algorithm 1. `nextWord( $w, N$ )`

INPUT: A word $w = a_1a_2 \cdots a_n$ and an NFA N .

OUTPUT: Returns the next word in the n^{th} cross-section of $L(N)$ according to length-lexicographic order if it exists. Otherwise, returns NULL. Updates S for a potential subsequent call to `nextWord` or `minWord`.

```

FOR  $i \leftarrow n, \dots, 1$ 
   $S_{i-1} = \text{top}(S)$ 
   $R = \{v \in \cup_{q \in S_{i-1}, a \in \Sigma} \delta(q, a) \mid v \text{ is } (n-i)\text{-complete}\}$ 
   $A = \{a \in \Sigma \mid \cup_{q \in S_{i-1}} \delta(q, a) \cap R \neq \emptyset\}$ 
  IF for all  $a \in A, a \leq a_i$ 
    pop( $S$ )
  ELSE
     $b_i = \min\{a \in A \mid a > a_i\}$ 
     $S_i = \{v \in \cup_{q \in S_{i-1}} \delta(q, b_i) \mid v \text{ is } (n-i)\text{-complete}\}$ 
    IF  $i \neq n$ 

```

```

    push( $S, S_i$ )
     $w' = w[1 \dots i - 1] \cdot b_i \cdot \text{minWord}(n - i, N)$ 
    RETURN  $w'$ 
RETURN NULL

```

Algorithm 2. crossSection(n, N)INPUT: A nonnegative integer n and an NFA N .OUTPUT: Enumerates the n^{th} cross-section of $L(N)$.

```

 $S = \text{empty stack}$ 
push( $S, \{q_0\}$ )
 $w = \text{minWord}(n, N)$ 
WHILE  $w \neq \text{NULL}$ 
    visit  $w$ 
     $w = \text{nextWord}(w, N)$ 

```

The algorithms `nextWord` and `crossSection` can be used in conjunction with any algorithms for `minWord` and for determining if a state is i -complete.

To get an enumeration algorithm, find the minimal word in each cross-section and call `nextWord` to get the rest of the words in the cross-section, until the required number of words is found.

Algorithm 3. enum(m, N)INPUT: A nonnegative integer m and an NFA N .OUTPUT: Enumerates the first m words accepted by N according to length-lexicographic order, if there are at least m words. Otherwise, enumerates all words accepted by N .

```

 $i = 0$ 
numCEC = 0
len = 0
WHILE  $i < m$  AND numCEC <  $s$  DO
     $S = \text{empty stack}$ 
    push( $S, \{q_0\}$ )
     $w = \text{minWord}(\text{len}, N)$ 
    IF  $w = \text{NULL}$ 
        numCEC = numCEC+1
    ELSE
        numCEC = 0
        WHILE  $w \neq \text{NULL}$  AND  $i < m$ 
            visit  $w$ 
             $w = \text{nextWord}(w, N)$ 
             $i = i + 1$ 
        len = len+1

```

The variable `numCEC` counts the number of consecutive empty cross-sections. If the count ever hits s , the number of states in N , then all the words accepted by N have been visited [1].

4 New Enumeration Algorithms

We present three sets of algorithms. Our algorithms improve on the asymptotic time over previous algorithms for the min-word and cross-section problems. In addition, for all of the enumeration problems discussed (namely min-word, cross-section, and enumeration), at least one of our algorithms has better practical running time than previous algorithms.

4.1 Ackerman-Mäkinen Sorted Algorithms

We present a modification of minWordM that runs in time linear in n , the length of the words in the cross-section. The original algorithm appears in [5]. Mäkinen's original algorithm is linear in n in the unit-cost model, and as shown in [1], the algorithm is quadratic in n in the bit complexity model. Here we present a modification of minWordM which is linear in n in the bit-complexity model. We also present a cross-section enumeration algorithm using this modification of minWordM that is linear in n . This algorithm is more efficient than the Lookahead-Matrix cross-section algorithm in terms of the number of states in the NFA, without compromising efficiency in any of the other parameters.

We now describe the setup for the modification of Mäkinen's algorithm. The algorithm builds a table for each state, representing the minimal words of lengths 1 through n that can be accepted from that state. In particular, $A^{min}[1]$ stores the minimal character that occurs on a transition from A to a final state; if no such transitions exists, then $A^{min}[1]$ is NULL. Index i in table A^{min} stores a pair (a, B) , with symbol a and state B , such that the minimal word of length i that occurs on a path from A to a final state is a concatenated with the minimal word of length $i - 1$ appearing on a path from state B to a final state; if no such path exists, then $A^{min}[i]$ is NULL.

We define an order on the set $\{A^{min}[i] \mid A \in Q\}$ as follows. If $A^{min}[i] = \text{NULL}$ and $B^{min}[i] \neq \text{NULL}$, then $A^{min}[i] < B^{min}[i]$. In addition, if $A^{min}[1] = a$ and $B^{min}[1] = b$ where $a < b$, then $A^{min}[1] < B^{min}[1]$. For $i \geq 2$, if $A^{min}[i] = (a, A')$ and $B^{min}[i] = (b, B')$ where $a < b$, or $a = b$ and $A'^{min}[i - 1] < B'^{min}[i - 1]$, then $A^{min}[i] < B^{min}[i]$.

Algorithm 4. minWordAMSorted(n, N)

INPUT: A positive integer n and an NFA N .

OUTPUT: Table $A^{min}[1, \dots, n]$ for each state $A \in Q$ where $A^{min}[i] = (a, B)$ and the minimal word of length i that occurs on a path from A to a final state is a concatenated with the minimal word of length $i - 1$ appearing on a path from state B to a final state.

```

FOR each  $A \in Q$ 
  IF for all  $a \in \Sigma$ ,  $\delta(A, a) \cap F = \emptyset$ 
     $A^{min}[1] = \text{NULL}$ 
  ELSE
     $A^{min}[1] = \min\{a \in \Sigma \mid \delta(A, a) \cap F \neq \emptyset\}$ 
FOR  $i \leftarrow 2, \dots, n$ 
  Sort the set  $\{A^{min}[i - 1] \mid A \in Q\}$ 

```

```

FOR each  $A \in Q$ 
   $a = \min\{a \in \Sigma \mid B^{min}[i-1] \neq \text{NULL}, B \in \delta(A, a)\}$ 
   $min = \text{NULL}$ 
  FOR each  $B \in Q$  such that  $B \in \delta(A, a)$ 
    IF  $B^{min}[i-1] \neq \text{NULL}$ 
      IF  $B^{min}[i-1] < min$  OR  $min = \text{NULL}$ 
         $min \leftarrow B^{min}[i-1]$ 
   $A^{min}[i] = (a, min)$ 
RETURN  $\{A^{min} \mid A \in Q\}$ 

```

Notice that sorting $\{A^{min}[i-1] \mid A \in Q\}$ takes $s \log s$ operations, since the sorting consists of sorting the pairs by their values of $\{A^{min}[i-2] \mid A \in Q\}$, for which the order has already been computed, followed by sorting the symbols in the pairs.

Since we assume that the characters on the transitions between states are sorted, `minWordAMSorted` uses $O(n(s \log s + d))$ operations. Observe also that `minWordAMSorted` uses $O(sn)$ space.

Theorem 1. *The algorithm `minWordAMSorted` uses $O((s \log s + d)n)$ operations and $O(sn)$ space to find the minimal word of length n accepted by an NFA, where s is the number of states and d is the number of transitions in the NFA.*

There are two main differences between this version of Mäkinen's algorithm and the original. The first is the mode of storage, as the original algorithm stored the minimal word of length i that occurs on a path from A to a final state in cell $A^{min}[i]$. The second modification is the sorting of the set $\{A^{min}[i] \mid A \in S\}$. These changes eliminate the need for expensive comparisons.

We can use `minWordAMSorted` as part of a cross-section enumeration algorithm, by replacing `minWord` with `minWordAMSorted`, giving algorithm `crossSectionAMSorted`. To determine if a state A is i -complete, we simply check whether $A^{min}[i] \neq \text{NULL}$. The algorithm `crossSectionAMSorted` finds the minimal word in the cross-section in $O((s \log s + d)n)$ operations, and finds the remaining words in $O(dt)$ where t is the output size. Therefore, `crossSectionAMSorted` uses $O(ns \log s + dt)$ operations.

Theorem 2. *The algorithm `crossSectionAMSorted` uses $O(ns \log s + dt)$ operations to enumerate the n^{th} cross-section of a given NFA, where s is the number of states and d is the number of transitions in the NFA and t is the number of characters in the output.*

We can also use `minWordAMSorted` as part of an enumeration algorithm, by replacing `minWord` with `minWordAMSorted` giving algorithm `enumAMSorted`. After enumerating the n^{th} cross-section we have a table $A^{min}[1, \dots, n]$ for each state A . To improve the performance of `enumAMSorted`, when `minWordAMSorted` is called for $n+1$, we reuse these tables, extending the tables by index $n+1$. Therefore, each call to `minWordAMSorted`(i, S) costs $O(s \log s + d)$. Finding the rest of the words in the cross-section costs $O(dt)$.

For each empty cross-section, the algorithm does $O(s \log s + d)$ operations. Therefore, `enumAMSorted` uses $O(c(s \log s + d) + dt)$ operations, where c is the number of cross-sections encountered through the enumeration. The running time is independent of alphabet size since the characters on transitions between every pair of states are sorted, so for every state on the state stack we can keep a pointer to the last characters used, and progress it when the state is revisited. When a state is removed from the state stack then the pointer is reset.

Theorem 3. *The algorithm `enumAMSorted` uses $O(c(s \log s + d) + dt)$ operations, where c is the number of cross-sections encountered throughout the enumeration.*

4.2 Ackerman-Mäkinen Boolean Algorithms

We introduce another modification of Mäkinen's algorithms which yields algorithms that are efficient both theoretically and practically. These algorithms are more efficient than the `AMSorted` algorithms in terms of the number of states in the NFA, without compromising efficiency in any of the other parameters. The min-word and cross-section algorithms in this section have better asymptotic running times than previous algorithms for these problems. In addition, the cross-section and enumeration algorithms presented in this section have the best practical running times.

The main function of the lookup tables constructed by Mäkinen's algorithm is to enable quick evaluation of i -completeness. Instead of storing characters, we propose storing boolean values in the lookup table. That is, $A^{min}[i]$ is true if and only if state A is i -complete.

Algorithm 5. `preprocessingAMBoolean`(n, N)

INPUT: A positive integer n and an NFA N .

OUTPUT: Table $A^{min}[1, \dots, n]$ for each state $A \in Q$ where $A^{min}[i]$ is true if and only if A is i -complete.

```

FOR each  $A \in Q$ 
  IF for all  $a \in \Sigma$ ,  $\delta(A, a) \cap F = \emptyset$ 
     $A^{min}[1] = \text{FALSE}$ 
  ELSE
     $A^{min}[1] = \text{TRUE}$ 
FOR  $i \leftarrow 2, \dots, n$ 
  FOR each  $A \in Q$ 
     $A^{min}[i] = \text{FALSE}$ 
    FOR each  $B \in \delta(A, a)$  for any  $a \in \Sigma$ 
      IF  $B^{min}[i-1] = \text{TRUE}$ 
         $A^{min}[i] = \text{TRUE}$ 
RETURN  $\{A^{min} \mid A \in Q\}$ 

```

The preprocessing takes $O((s + d)n)$ operations. Now, to find the minimal word of a given length n as well as the rest of the words in the cross-section,

we use the state stack (as described in Section 3), which takes $O(dt)$. This gives a $O(d(n+t))$ algorithm for cross-section enumeration (recall that we assume that the graph induced by the NFA is connected, and thus $s \leq d$.) We call this algorithm `crossSectionAMBoolean`.

Theorem 4. *The algorithm `crossSectionAMBoolean` uses $O(d(n+t))$ operations to enumerate the n^{th} cross-section, where d is the number of transitions in the NFA and t is the number of characters in the output.*

Similarly, to use this approach for enumerating the first m words accepted by an NFA, we extend the tables by one for each consecutive cross-section in $O(s+d)$ operations, and use the state stack to enumerate each cross-section. This gives a $O(d(c+t))$ time algorithm, where c is the number of cross-sections encountered throughout the enumeration. Or, since each non-empty cross-section adds at least one to t , we can express the asymptotic running time in terms of e , the number of empty cross-sections encountered by the algorithm, giving $O(d(e+t))$.

Theorem 5. *The algorithm `enumAMBoolean` uses $O(d(e+t))$ operations, where c is the number of empty cross-sections found throughout the enumeration, d is the number of transitions in the NFA, and t is the number of characters in the output.*

Note also, that we can terminate `crossSectionAMBoolean` after the first word is found, giving a $O(dn)$ running time algorithm for min-word - we call this algorithm `minWordAMBoolean`.

4.3 Intersection Cross-Section Enumeration Algorithm

We introduce a new cross-section enumeration algorithm, that is both very simple and has good asymptotic running time. The main idea of the algorithm is to create an NFA, such that when `crossSectionEnum` traverses the NFA looking for i -complete states, all reachable states are i -complete.

Let All_n be the minimal DFA that accepts all words over Σ^n .

Algorithm 6. `minWordIntersection`(n, N)

INPUT: A positive integer n and an NFA N .

OUTPUT: The minimal word of length n accepted by N .

1. Find NFA $C = N \times All_n$.
2. Perform a breadth-first-search starting from the final states of C using reverse transitions, and remove all unvisited states.
3. Find the minimal word of length n accepted by C by traversing C from the start state following minimal transitions.

The NFA C , the cross-product of N with the NFA that accepts all words of length n , can be constructed by concatenating n copies of N . Notice that C has $O(dn)$ transitions. Step 2 and 3 run in time proportional to the size of C . Therefore, `minWordIntersection` uses $O(dn)$ operations.

Theorem 6. *The algorithm `minWordIntersection` uses $O(dn)$ to find the minimal word of length n accepted by a given NFA, where d is the number of transitions in the NFA.*

To find all the words in the cross-section, perform breadth-first search, recording all the words of length n occurring on paths of length n starting from the start state. That is, if we store the sets of states visited by `minWordIntersection` on a state stack, and use `minWordIntersection` with `enumCrossSection`, we get algorithm `crossSectionIntersection`.

We can use `minWordIntersection` to make a cross-section enumeration algorithm, `crossSectionIntersection`. That is, we can use the `enumCrossSection` algorithm on C instead of N , with `minWordIntersection`. All paths starting from the start state in C lead to a final state in n states; therefore, testing for i -completeness is not needed.

The algorithm `crossSectionIntersection` uses $O(dn)$ operations to find the minimal word of length n , and $O(dt)$ to find every other character in the cross-section. Therefore, `crossSectionIntersection` uses $O(d(n+t))$ operations.

Theorem 7. *The algorithm `crossSectionIntersection` uses $O(d(n+t))$ operations to enumerate the n^{th} cross-section in a given NFA, where d is the number of transitions in the NFA, and t is the number of characters in the output.*

5 Experimental Results

We compare the practical performance of the new algorithms with the best previously known algorithms. From [1], we know that among the Grail enumeration algorithms, the lookahead-matrix algorithms, and two versions of Mäkinen's algorithm, the Mäkinen algorithms tend to have the best practical performance. In some cases, Lookahead-Matrix performs better than Mäkinen's algorithm [1]. Therefore, we compare the new algorithms presented here with Mäkinen's and the lookahead-matrix algorithms.

A large body of tests was randomly generated. Most tests follow the following format: 100 NFAs were randomly generated with a bound on the number of vertices and alphabet size. The probability of placing an edge between any two states was randomly generated. The probability of any state being final or the number of final states was randomly generated within a specified range. The algorithms were tested on NFAs with differing number of states, varying edge densities, various alphabet sizes, and different proportions of final states.

In addition, we tested the algorithms on NFAs on which `crossSectionLM` performs better than `crossSectionM`. An example of this type of NFAs is presented in Figure 1. To see why `crossSectionM` has quadratic running time on this type of NFAs, see [1]. All but the tests on the L_m NFAs (see Figure 1) are on sets of randomly generated NFAs with up to 10 nodes.

We perform three groups of tests. The first group is aimed at determining the practical efficiency of the new min-word algorithms. We compared `minWordM`, `minWordAMSorted`, `minWordAMBoolean`, and `minWordLM`. In most of these tests,

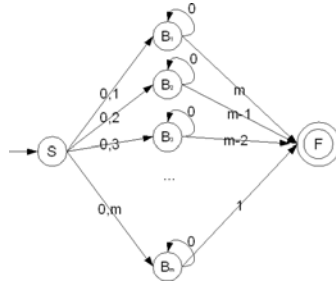


Fig. 1. NFA L_m

minWordAMSorted algorithm performs best. The only tests in which minWordAMSorted did not have the best performance was in some of the tests with the L_m NFAs (see Figure 1), but even on these NFAs this algorithm’s performance was close to the performance of the fastest algorithm for these tests.

The second set of tests evaluates the new cross-section enumeration algorithms. We compare `crossSectionM`, `crossSectionAMSorted`, `crossSectionAMBoolean`, `crossSectionLM`, and `crossSectionIntersection`. We found that, in general, `crossSectionAMBoolean` and `crossSectionM` perform better than the other algorithms, with their performance very close to each other. On NFAs on which `crossSectionM` is quadratic in the size of the cross-section (the L_m automata), `crossSectionAMBoolean` outperforms the other algorithms. Thus, `crossSectionAMBoolean` has the overall best asymptotic and practical running time. In addition, we found that `crossSectionIntersection` is significantly slower than the other algorithms.

In the third set of tests we compared the practical performance of `enumM`, `enumAMSorted`, `enumAMBoolean`, and `enumLM`. The algorithm `enumAMBoolean` has the best running time. In a few cases, `enumAMBoolean` was outperformed by a very small amount by `enumM` or `enumAMSorted`. Due to the nature of the algorithms, it appears that the cases when `enumAMBoolean` is outperformed can be attributed to language or implementation specific details. Thus, the `AMBoolean` algorithms have the best practical performance for both the cross-section and the enumeration problem.

The tests were written in C# 3.0 and run on Microsoft Windows Vista Business, Intel(R) Core(TM)2 Duo CPU 1.80GHz, 1.96 GB of RAM. We summarize our experiments in Tables 2-4 in the appendix.

6 Summary and Future Work

We presented three sets of algorithms: The AM-Sorted min-word, cross-section and enumeration algorithms, the AM-Boolean min-word, cross-section and enumeration algorithms, and the intersection-based min-word and cross-section algorithms. We then compared the practical performance of these algorithms

with the two best previously known sets of algorithms, lookahead-matrix and Mäkinen. We found that `minWordAMSorted`, `crossSectionAMBoolean`, and `enumAMBoolean` have the best practical running times.

Table 1 summarizes the asymptotic running times of the new and old algorithms. Recall that s is the number of states and d is the number of transitions in the NFA, t is the number of characters in the output, c is the number of cross-sections encountered throughout the enumeration, and e is the number of empty cross-sections encountered throughout the enumeration.

Table 1. Asymptotic performances of the algorithms

	AMSorted	AMBoolean	Intersection	Mäkinen	lookahead-matrix
Min-word	$O((s \log s + d)n)$	$O(dn)$	$O(dn)$	$O(s^2 n^2)$	$O(s^{2.376} n)$
Cross-section	$O(ns \log s + dt)$	$O(d(n + t))$	$O(d(n + t))$	$O(d(n^2 + t))$	$O(s^{2.376} n + dt)$
Enumeration	$O(c(s \log s + d) + dt)$	$O(d(e + t))$	x	$O(d(e + t))$	$O(s^{2.376} c + dt)$

This shows that $O(dn)$ is the best asymptotic running time for the min-word problem, $O(d(n + t))$ is the best running time for the cross-section problem, and $O(d(e + t))$ is the best running time for the enumeration problem - all achieved by the AMBoolean algorithms. It would be interesting to try to find more efficient algorithms for these problems, or explore the question of whether these are optimal by proving lower bounds.

Acknowledgments

We would like to thank Moshe Vardi for a very helpful discussion in which he proposed the idea behind the intersection-based cross-section algorithm.

References

1. Ackerman, M., Shallit, J.: Efficient enumeration of words in regular languages. Theoretical Computer Science. Elsevier (2009)
2. Anderson, T., Rampersad, N., Santean, N., Shallit, J.: Finite automata, palindromes, patterns, and borders. In: CoRR, abs/0711.3183 (2007)
3. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, London (1971)
4. University of Western Ontario, Department of Computer Science: Grail+ (December 2008), <http://www.csd.uwo.ca/Research/grail/index.html>
5. Mäkinen, E.: On lexicographic enumeration of regular and context-free languages. Acta Cybernet 13, 55–61 (1997)

Appendix

We summarize our experiments, recording time measured in seconds, in Tables 2–4. When an “x” appears, the test case has not been run to completion due to a high running time.

Table 2. The performances of the Min-word algorithms

	n	Mäkinen	AMSorted	AMBoolean	Lookahead-Matrix
L_2	5000	3.120	0.031	0.031	0.031
L_2	50,000	x	2.777	3.229	3.307
L_2	10,000	x	18.283	35.864	29.047
L_9	2000	4.336	0.0468	0.0156	0.0780
L_9	20000	x	0.624	0.468	0.889
L_9	50,000	x	3.4164	3.7908	4.0872
L_9	80,000	x	18.782	17.257	14.6016
L_9	100,000	x	18.938	36.722	28.018
L_9	120,000	x	47.252	50.575	42.713
Alp. size 2, ≤ 10 nodes	3	1.045	0.8976666	1.6160000	2.7340000
Alp. size 2, 1 final	5	0.083	0.078	0.109	0.114
Alp. size 2, 1 final	7	0.114	0.088	4.066	5.699
Alp. size 2, 1 final	8	0.114	0.099	36.462	48.599
Alp. size 3, 1 final	7	0.047	0.042	3.6244000	4.8932000
Alp. size 3, 1 final	8	0.078	0.042	33.285	43.597
Alp. size 3, 1 final	500	42.713	0.905	x	x
Alp. size 15, 1 final	100	2.002	0.608	x	x
Alp. size 15, 1 final	500	40.092	2.938	x	x
Alp. size 5	200	8.611	0.619	x	x
Alp. size 10	200	6.839	0.816	x	x
Alp. size 10	500	39.359	1.950	x	x
Alp. size 10	600	57.164	2.335	x	x

Table 3. The performances of the Cross-section algorithms

	n	Mäkinen	AMSorted	AMBoolean	Lookahead-Matrix	Intersection
L_2	1000	0.153	0.009	0.006	0.008	x
L_2	5000	3.617	0.092	0.057	0.063	x
L_2	10000	14.191	0.217	0.157	0.183	x
L_9	1000	1.15	0.045	0.026	0.053	x
L_9	5000	28.371	0.387	0.307	0.413	x
Alp. size 2, dense graph	5	1.576	1.538	1.539	1.722	21.461
Alp. size 2, dense graph	6	24.163	24.597	24.424	28.126	x
Alp. size 2, dense graph	7	0.191	0.197	0.186	0.226	1.782
Alp. size 2, sparse graph	10	10.322	10.385	10.379	12.908	2:45.835
Alp. size 2, all final	5	1.630	1.632	1.617	1.748	41.961
Alp. size 10, sparse graph	7	1.591	1.580	1.602	1.714	11.302

Table 4. The performances of the Enumeration algorithms

	n	Mäkinen	AMSorted	AMBoolean	Lookahead-Matrix
L_2	100	0.003	0.002	0.002	0.003
L_2	1000	0.181	0.161	0.162	0.210
L_2	10000	38.681	37.286	36.896	49.048
L_9	100	0.004	0.001	0.001	0.001
L_9	1000	0.034	0.029	0.025	0.037
L_9	10000	2.999	2.702	2.678	0.3461
L_9	20000	14.635	12.751	13.027	16.221
Alp. size 2	10	3.966	3.796	3.262	7.677
Alp. size 2	12	5.085	4.882	4.575	8.995
Alp. size 2, dense graph	12	0.046	0.041	0.036	0.047
Alp. size 2, dense graph	20	0.089	0.102	0.083	0.089
Alp. size 2, dense graph	30	0.159	0.159	0.152	0.167
Alp. size 2, dense graph	100	15.882	16.487	16.589	18.477
Alp. size 2, sparse graph	10	0.031	0.028	0.023	0.029
Alp. size 2, sparse graph	20	0.036	0.034	0.028	0.037
Alp. size 2, sparse graph	30	0.041	0.041	0.036	0.044
Alp. size 2, sparse graph	100	0.155	0.146	0.131	0.172
Alp. size 2, sparse graph	1000	12.55	12.504	12.137	13.944
Alp. size 2, all final.	10	0.039	0.038	0.032	0.037
Alp. size 2, all final.	15	0.049	0.052	0.043	0.05
Alp. size 2, all final.	20	0.1	0.096	0.086	0.098
Alp. size 2, all final.	30	2.511	2.497	2.426	3.588
Alp. size 10, sparse graph	100	0.064	0.062	0.058	0.066
Alp. size 2, sparse graph	1000	2.843	2.844	2.862	3.082
Alp. size 2, sparse graph	2000	17.010	17.163	17.685	17.264

Convex Partitions with 2-Edge Connected Dual Graphs

Marwan Al-Jube¹, Michael Hoffmann², Mashhood Ishaque¹,
Diane L. Souvaine¹, and Csaba D. Tóth³

¹ Department of Computer Science, Tufts University, Medford, MA, USA*
{maljub01,mishaq01,dls}@cs.tufts.edu

² Institute of Theoretical Computer Science, ETH Zürich, Switzerland
hoffmann@inf.ethz.ch

³ Department of Mathematics, University of Calgary, AB, Canada**
cdtoth@ucalgary.ca

Abstract. It is shown that for every finite set of disjoint convex polygonal obstacles in the plane, with a total of n vertices, the free space around the obstacles can be partitioned into open convex cells whose dual graph (defined below) is 2-edge connected. Intuitively, every edge of the dual graph corresponds to a pair of adjacent cells that are both incident to the same vertex.

Aichholzer *et al.* recently conjectured that given an even number of line-segment obstacles, one can construct a convex partition by successively extending the segments along their supporting lines such that the dual graph is the union of two edge-disjoint spanning trees. Here we present a counterexamples to this conjecture, with n disjoint line segments for any $n \geq 15$, such that the dual graph of any convex partition constructed by this method has a *bridge* edge, and thus the dual graph cannot be partitioned into two spanning trees.

1 Introduction

For a finite set S of disjoint convex polygonal obstacles in the plane $\mathbb{R}^2$, a *convex partition* of the free space $\mathbb{R}^2 \setminus (\bigcup S)$ is a set C of open convex regions (called *cells*) such that the cells are pairwise disjoint and their closures cover the entire free space. Since every vertex of an obstacle is a reflex vertex of the free space, it must be incident to at least two cells. Let σ be an assignment of every vertex to two adjacent convex cells in C . A convex partition C and an assignment σ define a *dual graph* $D(C, \sigma)$: the cells in C correspond to the nodes of the dual graph, and each vertex v of an obstacle corresponds to an edge between the two cells assigned to v (see Fig. 1). Double edges are possible, corresponding to two endpoints of a line-segment obstacle on the common boundary of two cells.

* Partially supported by NSF grants CCF-0431027 and CCF-0830734, and Tufts Provost's Summer Scholars Program.

** Supported by NSERC grant RGPIN 35586. Research done at Tufts University.

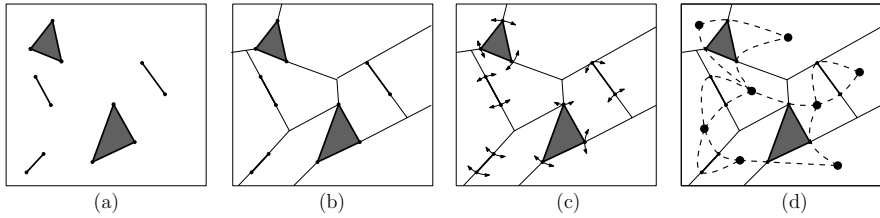


Fig. 1. (a) Five obstacles with a total of 12 vertices. (b) A convex partition. (c) An assignment σ . (d) The resulting dual graph.

It is straightforward to construct an arbitrary convex partition for a set of convex polygons as follows. Let V denote the set of vertices of the obstacles; and let π be a permutation on V . Process the vertices in the order π . For a vertex $v \in V$, draw a directed line segment (called *extension*) that starts from the vertex along the angle bisector (for a line-segment obstacle, the extension is collinear with the obstacle), and ends where it hits another obstacle, a previous extension, or infinity (the bounding box). For k convex obstacles with a total of n vertices, this naïve algorithm produces a convex partition with $n - k + 1$ cells, if no two extensions are collinear. For example, for n disjoint line segments (with $2n$ endpoints) in general position, we obtain $n + 1$ cells. If the obstacles are in general position, then each vertex v is incident to exactly two cells, lying on opposite sides of the extension emanating from v . Hence the assignment σ is unique, and the choice of permutation π completely determines the dual graph $D(\pi)$. We call this a **STRAIGHT-FORWARD** convex partition, and a **STRAIGHT-FORWARD** dual graph, which depends only on the permutation π of the vertices.

Our Results. We show instances where **no** permutation π produces a **STRAIGHT-FORWARD** dual graph $D(\pi)$ that is 2-edge connected (Section 2). This is a counterexample to a conjecture by Aichholzer *et al.* [1].

We show that for every finite set of disjoint convex polygons in the plane there is a convex partition (not necessarily **STRAIGHT-FORWARD**) and an assignment that produces a 2-edge connected dual graph (Section 3).

Motivation. A *plane matching* is a set of n disjoint line segments in the plane, which is a perfect matching on the $2n$ endpoints. Two plane matchings on the same vertex set are *compatible* if there are no two edges that cross, and are *disjoint* if there is no shared edge. Aichholzer *et al.* [1] conjectured that for every plane matching on $4n$ vertices, there is a disjoint compatible plane matching. (*compatible geometric matchings conjecture*). They proved that their conjecture holds if the $2n$ segments in the matching admit a convex partition whose dual graph is the union of two edge-disjoint spanning trees, and the two endpoints of each segment corresponds to distinct spanning trees. Aichholzer *et al.* further conjectured for the $4n$ endpoints of $2n$ line segments in the plane, there is a permutation π such that $D(\pi)$ is the union of two edge-disjoint spanning trees (*two spanning trees conjecture*).

The conjecture would immediately imply that such a dual graph is 2-edge connected. Benbernou *et al.* [4] claimed that there is always a permutation π such that $D(\pi)$ is 2-edge connected—but there was a flaw in their argument [5]. Our first result shows that such permutation π does **not** always exist, and it also refutes the *two spanning trees conjecture* of Aichholzer *et al.* [1].

Related Work. Given a set of convex polygonal obstacles and a bounding box, we may think of the bounding box as a simple polygon and the obstacles as polygonal holes. Then the problem of creating a convex partition becomes that of decomposing the simple polygon with holes into convex parts. Convex polygonal decomposition has received considerable attention in the field of computational geometry. The focus has been to produce a decomposition with as few convex parts as possible. Lingas [14] showed that finding the *minimal convex decomposition* (decomposing the polygon into the fewest number of convex parts) is NP-hard for polygons with holes. However, for polygons without holes, minimal convex decompositions can be computed in polynomial time [8,11]—see [10] for a survey on polygonal decomposition.

While minimal convex decomposition is desirable, it is not the only criterion for the *goodness* of a convex partition (decomposition). In fact, the measure of the quality of a convex partition can be specific to the application domain. In Lien’s and Amato’s work on approximate convex decomposition [13] with applications in skeleton extraction, the goal is to produce an approximate (not all cells are convex) convex partition that highlights salient features. In the *equitable* convex partitioning problem, all convex cells are required to have the same value of some measure e.g. the same number of red and blues points [9], or the same area [7].

2 Counterexample for Two Spanning Trees Conjecture

Theorem 1. *For every $n \geq 15$, there are n disjoint line segments in the plane such that the dual graph $D(\pi)$ has a bridge edge for every permutation π .*

Proof. We show that for the 15 line segments in Fig. 2, every permutation π produces a STRAIGHT-FORWARD dual graph $D(\pi)$ with a bridge edge (removing this edge disconnects the dual graph). Our construction consists of three rotationally symmetric copies of a configuration with 5 segments $\{A_1, A_2, \dots, A_5\}$, which we call a *star structure*. We can generate larger constructions by adding segments whose supporting lines avoid the convex hull of this configuration.

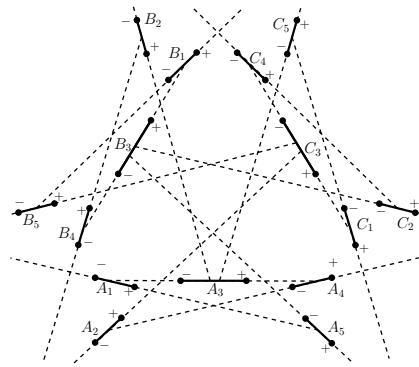


Fig. 2. Counterexample with $n = 15$

In Fig. 2 the dotted lines represent the arrangement of all possible extensions of the given line segments. Denote the right endpoint of a segment by ‘+’ and the left endpoint by ‘-’. The set of all possible permutations can be described in terms of only two cases by focusing on the star structure A .

Case 1. The extensions from endpoints A_3^- and A_3^+ hit the segments A_1 and A_4 , respectively; i.e. the extensions from endpoints A_2^+ and A_5^- terminate either at the extensions from endpoints A_3^- and A_3^+ , respectively or earlier (Fig. 3). It can be easily verified that in this case every permutation of the four endpoints $\{A_1^+, A_2^+, A_4^-, A_5^-\}$ produces a bridge in the dual graph. The same reasoning applies to the structures B and C because of symmetry.

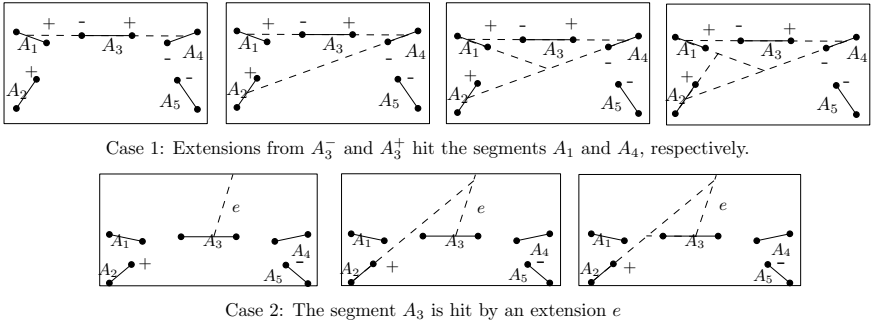


Fig. 3. Permutations for the counterexample

Case 2. Therefore, to avoid a bridge edge in the dual graph, there must be at least one endpoint in each star structure whose extension goes beyond the structure. Since when two extensions meet in a STRAIGHT-FORWARD convex partition, one of the extensions must continue in a straight line, at least one of these three endpoints will have its extension hit a segment (A_3 , B_3 or C_3) in a different structure. Assume w.l.o.g segment A_3 is hit by an extension e from either B_2^+ or C_5^- . Then an extension from either A_2^+ or A_5^- hits e , which together with A_3^- or A_3^+ creates a bridge in the dual graph. $\square$

3 Constructing a Convex Partition

We showed in Section 2 that in some instances, no STRAIGHT-FORWARD dual graph is 2-edge connected. In this section we present an algorithm that produces a convex partition with a 2-edge connected dual graph. We will start from an arbitrary STRAIGHT-FORWARD convex partition, and apply a sequence of *local modifications*, if necessary, until the dual graph becomes 2-edge connected. Our local modifications will not change the number of cells. We define a class of convex partitions (DIRECTED-FOREST) that includes all STRAIGHT-FORWARD convex partitions and is closed under the local modifications we propose.

The basis for local modifications is a simple idea. In a STRAIGHT-FORWARD convex partition, extensions are created sequentially (each vertex emits a directed ray) and whenever two directed extensions meet at a *Steiner* vertex v (defined below), the earlier extension continues in its original direction, and the later one terminates. Here, however, we allow the two directed extensions to merge and continue as one edge in any direction that maintains the convexity of all the angles incident to v (Fig. 4(a)). Merged extensions provide considerable flexibility.

Definition 1. For a given set S of disjoint obstacles, the class of DIRECTED-FOREST convex partitions is defined as follows (refer to Fig. 4): The free space $\mathbb{R}^2 \setminus (\bigcup S)$ is partitioned into convex cells by directed edges (including directed rays going to infinity). Each endpoint of a directed edge is either a vertex of S or a Steiner vertex (lying in the interior of the free space, or on the boundary of an obstacle). We require that

- every vertex in V (a vertex of an obstacle) emits exactly one outgoing edge;
- every Steiner point in the interior of the free space is incident to exactly one outgoing edge;
- no Steiner point on a convex obstacle is incident to any outgoing edge; and
- the directed edges do not form a cycle.

It is easy to see that a STRAIGHT-FORWARD convex partition belongs to the DIRECTED-FOREST class.

Proposition 1. There is an obstacle vertex on the boundary of every cell.

Proof. Consider a directed edge on the boundary of a cell. Follow directed edges in reverse orientation along the boundary. Since directed edges cannot form a cycle, and the out-degree of every Steiner vertex is at most one, there must be at least one obstacle vertex on the boundary of the cell. $\square$

In a DIRECTED-FOREST, we can also follow directed edges (in forward direction) from any vertex in V to an obstacle or to infinity, since the out-degree of each vertex is always exactly one, unless the vertex lies on the boundary of an obstacle or at infinity. For connected components of extensions (directed edges), we use the concept of *extension trees* introduced by Bose *et al.* [6].

Definition 2. The **extended-path** of a vertex $v \in V$ is a directed path along directed edges starting from v and ending on an obstacle or at infinity. Its (relative) interior is disjoint from all obstacles.

Definition 3. An **extension tree** is the union of all extended-paths that end at the same point, which is called the **root** of the extension tree. The **size** of an extension tree is the number of extended-paths included in the tree.

A vertex $v \in V$ may be incident to more than two cells. It is incident to $\ell + 2$ cells if it is incident to ℓ incoming edges. In our construction, we let σ assign a vertex v of an obstacle to the two cells adjacent to the unique outgoing edge incident to v . With this convention, a bridge edge in the dual graphs $D(C, \sigma)$ can be characterized by a forbidden pattern (see Fig. 4(b)).

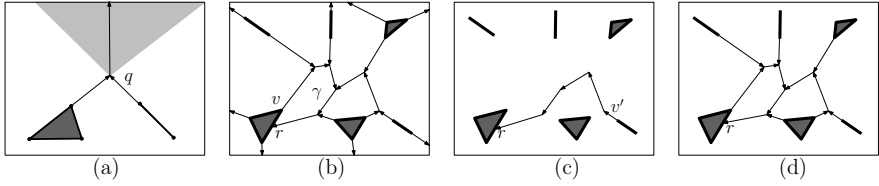


Fig. 4. (a) If two incoming extensions meet at q , the merged extension may continue in any direction within the opposing wedge. (b) A convex partition formed by directed line segments. The extended path γ originates at v and terminates at r , two points on the same obstacle. The edge at v is a bridge in the dual graph, and γ is called *forbidden*. (c) A single extended-path emitted by v' . (d) A single extension tree rooted at r .

Definition 4. An extended-path starting at $v \in V$ is called **forbidden** if it ends at the obstacle incident to v . A **forbidden** extended-path, together with the boundary of the incident obstacle, forms a simple closed curve, which encloses a bounded region.

Lemma 1. A dual graph $D(C, \sigma)$ of a DIRECTED-FOREST convex partition is 2-edge connected if and only if no vertex $v \in V$ emits a forbidden extended-path.

Proof. First we show that a forbidden extended-path implies a bridge in the dual graph. Let γ be a forbidden extended-path, starting from vertex v of an obstacle, and ending at point r on the boundary of the same obstacle (see Figures 4(b), 5, 6). Extended-path γ together with the obstacle boundary between v and r forms a simple closed curve and partitions the free space into two regions R_1 and R_2 , each of which is the union of some convex cells. Let V_1 and V_2 be the set of nodes in the dual graph corresponding to the convex cells in these regions, respectively. Point v is the only obstacle vertex along γ . If an edge e of the dual graph connects some node in V_1 to a node in V_2 , then e corresponds to a vertex of an obstacle whose unique outgoing edge is part of γ . But v is the only such vertex. This implies that there is a bridge in the dual graph, whose removal disconnects V_1 from V_2 .

Next we show that a bridge in the dual graph implies a forbidden extended-path. Assume that V_1 and V_2 form a partition of V in $D(C, \sigma)$ such that V_1 and V_2 are connected by a bridge edge e . The two node sets correspond to two regions, R_1 and R_2 , in the free space. Let β be boundary separating the two regions. We first show that one of these regions is bounded.

Suppose for contradiction that both regions R_1 and R_2 are unbounded. Note that β must contain at least two directed edges of the convex partition that go to infinity. Since every Steiner vertex in the interior of the free space has an outgoing edge, β must contain at least two extended-paths. Hence β contains at least two vertices of some obstacles, and the adjacent outgoing edges. Thus there are at least two edges in the dual graph between the node sets V_1 and V_2 , therefore, e is not a bridge edge.

Now assume without loss of generality that the region R_1 is bounded, and thus the separating boundary β is a closed curve. If we pick an arbitrary directed extension along β and follow β in reverse direction, then we arrive to a segment endpoint v . Assume that v corresponds to the bridge edge e . Then we arrive to the same segment endpoint v starting from any directed extension along β . This means that all directed edges along β are in the extended-path of v . Since β is a closed curve, the extended-path of v must end on the boundary of the obstacle incident to v , and thus it a forbidden extended-path. $\square$

Corollary 1. *An extension tree with its root at infinity cannot contain a forbidden extended-path.*

3.1 Convex Partitioning Algorithm

We construct a convex partition as follows. We first create a STRAIGHT-FORWARD convex partition, which is in the class of DIRECTED-FOREST convex partitions. Let $\mathcal{T}$ denote the set of extension trees. Each extension tree may contain one or more forbidden extended-paths. If an extension tree $t \in \mathcal{T}$ contains a forbidden extended-path γ , then we continuously deform t with a sequence of local modifications until a vertex of an obstacle collides with the relative interior of t (subroutine FLEXTREE(t)). At that time, t splits into two extension trees t_1 and t_2 such that each of these two trees is strictly smaller in size than t . An extension tree of size one is a straight-line extension, and cannot contain a forbidden extended-path. Since the number of extended-paths is fixed (equal to the number of vertices in V), eventually no extension tree contains any forbidden extended-path, and we obtain a convex partition whose dual graph has no bridges by Lemma 1.

Algorithm 1. CREATECONVEXPARTITION(S)

Given: A set S of disjoint convex polygons having n vertices in total.

Create a STRAIGHT-FORWARD convex partition.

Let $\mathcal{T}$ be set of extension trees in the partition.

while there is an extension tree $t \in \mathcal{T}$ containing a forbidden extended-path **do**

FLEXTREE(t)

end while

Algorithm 2. FLEXTREE(t)

Let γ be a forbidden extended-path contained in t .

while γ is still a forbidden extended-path **do**

$(t, \gamma) = \text{EXPAND}(t, \gamma)$

end while

Let $v' \in V$ be a vertex of an obstacle where the extended-path γ now terminates.

Split tree t into two extension trees t_1 and t_2 . Subtree t_1 consists of the extended-paths that terminate at the original endpoint of γ . Subtree t_2 consists of the extended-paths that now terminate at v' .

For a finite set S of disjoint convex polygonal obstacles in the plane, the main loop of our partition algorithm is `CREATECONVEXPARTITION( $S$ )`. It calls subroutine `FLEXTREE( $t$ )` for every extension tree that contains a forbidden extended-path. `FLEXTREE( $t$ )`, in turn, calls subroutine `EXPAND( $t, \gamma$ )` for a forbidden extended path γ , as described in Section 3.2.

3.2 Local Modifications: `EXPAND( $t, \gamma$ )`

Consider a forbidden extended-path γ contained in an extension tree $t \in \mathcal{T}$. Path γ starts from a vertex $v \in V$, and ends at a root r lying on the boundary of the obstacle $s \in S$ incident to v . Path γ together with the portion of the boundary of s between v and r bounds a simple polygon P that does not contain s in its interior.

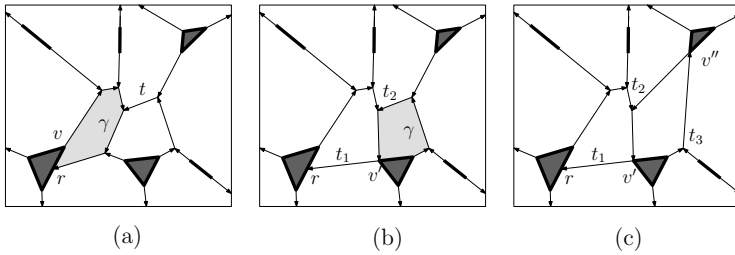


Fig. 5. (a) An extension tree t with a forbidden extended-path. (b) After deforming and splitting t into two trees, t_2 contains a forbidden extended-path. (c) Deforming and splitting t_2 eliminates all forbidden extended-paths.

We continuously deform the boundary of P , together with extension tree t , until it collides with a new vertex $v' \in V$ that is not incident to s . Similar continuous motion arguments have been used for proving combinatorial properties in [2,3,12]. We deform P in a sequence of local modifications, or *steps*. Each step involves two adjacent edges of the polygon P . The *vertices* of P are v, r and the Steiner points where P has an interior angle different from 180° . Steiner vertices where P has an interior angle of 180° are considered interior points of edges of P . Each step of the deformation will (i) increase the interior of the polygon P , (ii) keep r a vertex of P , and (iii) maintain a valid `DIRECTED-FOREST` convex partition. The third condition implies, in particular, that every cell has to remain convex. Also, since the interior of P is increasing, some cells in the exterior of P (and adjacent to P) will shrink—we ensure that all cells adjacent to P have a nonempty interior.

Where to perform a local deformation step? The polygon P is modified either at a convex vertex x on the convex hull of P or at a reflex vertex x' (with special properties). These vertices x and x' are calculated at the start of each local deformation step.

Consider the edge of the obstacle s that is incident to the point v , and is part of the boundary of the polygon P . Let ℓ be the supporting line through this

edge. The obstacle s lies completely in one of the closed halfplanes bounded by ℓ (since s is convex). Let x be a vertex of P furthest away from the supporting line ℓ in the other halfplane. Clearly, x is a convex vertex of P (interior angle less than 180°), otherwise it will not be the furthest. The goal is to expand the polygon P by modifying the edges xy and xz incident to x . Imagine grabbing the vertex x and pulling it away from the polygon P stretching the edges xy and xz . But this expansion can only occur if both the edge xy and xz are *flexible*. An edge of P is *inflexible* if there is a convex cell in the interior of P that has an angle of 180° at one of the two endpoints of the edge. Since x is a convex vertex, the edge xy or xz can be inflexible if and only if some convex cell has an angle of 180° at y or z , respectively (Fig. 6).

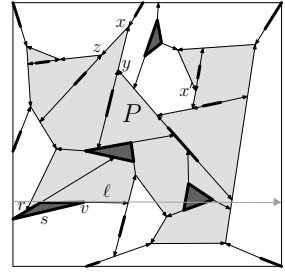


Fig. 6. Polygon P corresponding to a forbidden extended-path $v, \dots, r$; convex vertex x ; inflexible edges xy and xz ; reflex vertex x'

In the case when at least one of the edges incident to x is inflexible, local modification of P takes place at a reflex vertex x' . Assume w.l.o.g xy is inflexible. Then y must be a reflex vertex of P (every inflexible edge of P is incident to a reflex vertex). Starting with the reflex vertex y , move along the boundary of P in the direction away from x . Let x' be the first reflex vertex encountered such that one of the edges incident to x' is flexible. It is not difficult to verify that there is always one such vertex x' (Proposition 2).

Proposition 2. *If x is incident to an inflexible edge, then there is a reflex polygonal chain along P of length ≥ 1 that includes this inflexible edge and terminates at a reflex vertex x' of P that has exactly one flexible edge.* $\square$

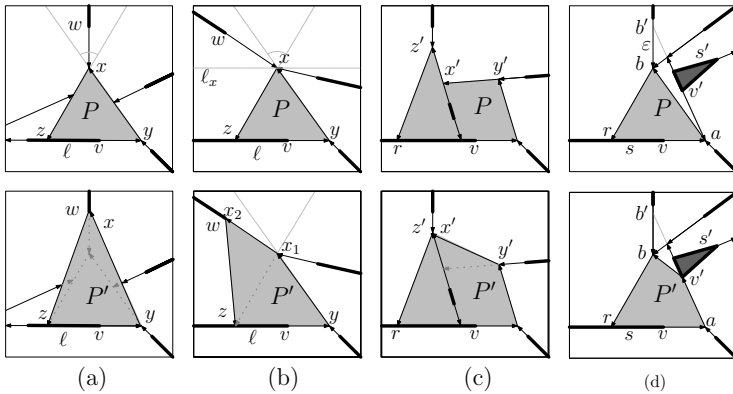


Fig. 7. Three local operations: (a) Convex vertex x , incoming edge w in the wedge. (b) Convex vertex x , no incoming edge in the wedge. (c) Reflex vertex x . (d) The case of a collapsing cell.

How to perform a local deformation step? Local deformation of P takes place either at a convex vertex x (Case 1 and 2), or at a reflex vertex x' (Case 3). Since the number of cells in the convex partition must remain the same, it is necessary to check for the collapse of a cell in the exterior of P (Case 4).

Case 1. Both edges xy and xz of P incident to x are flexible, and there is an edge wx in the opposing wedge of $\angle yxz$. Fig. 7(a). Then continuously move x along xw towards w while stretching the edges xy and xz .

Case 2. Both edges xy and xz of P incident to x , are flexible, and there is no edge in the opposing wedge of $\angle yxz$. Fig. 7(b). Let ℓ_x be a line parallel to ℓ passing through x , and let w be a neighbor of x on the opposite side of ℓ_x . Assume that z and w are on the same side of the angle bisector of $\angle yxz$. Then split x into two vertices x_1 and x_2 . Now x_1 remains fixed at x and x_2 moves continuously along xw towards w stretching the edge x_2z .

Case 3. At least one edge incident to x is inflexible; then there is a reflex vertex x' such that edge $x'z'$ is inflexible, and $x'y'$ is flexible. Fig. 7(c). Continuously move x' along $x'z'$ towards z' while stretching the edge $x'y'$.

Case 4. A further $\varepsilon > 0$ stretching of some edge ab to position ab' , where vertex b continuously moves along segment bb' , would collapse a cell in the exterior of P . Fig. 7(d). Then the triangle $\Delta abb'$ lies in the free space and ab' contains a side of an obstacle $s' \neq s$ (cf. Proposition 3 below). Let $v' \in ab'$ be the vertex of this obstacle that lies closer to a . Stretch edge ab of P into the path (a, v', b) .

When to stop a local deformation step? Continuously deform one or two edges of P , at either a convex vertex x or a reflex vertex x' , until one of the following conditions occurs:

- an angle of a convex cell interior to P or an angle of P becomes 180° ;
- two vertices of the polygon P collide;
- one of the edges of P collides either in its interior or at its endpoint with a vertex v of an obstacle;
- a further $\varepsilon > 0$ deformation would collapse a cell in the exterior of P .

Since a local deformation step does not always terminate in a collision with an obstacle vertex v' , the subroutine $\text{FLEXTREE}(t)$ decides at the end of each step whether more local modifications are needed.

3.3 Correctness of the Algorithm

We prove that we can eliminate all forbidden extended-paths and obtain a DIRECTED-FOREST convex partition with a 2-edge connected dual graph. Let t be a extension tree, containing a forbidden extended-path γ starting from $v \in V$ and ending at root r . First we show that in $\text{EXPAND}(t, \gamma)$, the four cases cover all possibilities.

Proposition 3. *If a further $\varepsilon > 0$ deformation of some edge ab to position ab' , where b continuously moves along segment bb' , would collapse a cell in the exterior of P , then the triangle $\Delta abb'$ lies in the free space and segment ab' contains a side of an obstacle $s' \neq s$.*

Proof. A continuous deformation of ab to ab' , where b' moves along segment bb' , sweeps triangle $\Delta abb'$. Hence the interior of this triangle cannot contain any obstacle. Assume that cell $c \in C$ would collapse if ab reaches position ab' . By Proposition 1, there is a vertex $v' \in V$ on the boundary of cell c , and so v' must lie on the segment ab' . Note that no extended-path can reach v' from the triangle $\Delta abb'$. Hence the only two edges along the boundary of c incident to v' are the extension emitted by a side of the obstacle s' containing v' . It follows that segment ab' contains a side of obstacle s' .

It remains to be shown that $s' \neq s$ (that is, v and v' are vertices of distinct obstacles). In Case 1–2, $b = x$ is the convex vertex of P that lies furthest from the supporting line ℓ , and b moves continuously away from ℓ . Therefore both b and b' are in the open halfplane bounded by ℓ , and so edge ab' cannot contain an edge of the obstacle s . In Case 3, $b = x'$ is a reflex vertex of P and it moves continuously along a reflex chain along the boundary of P between x' and x (c.f., Proposition 2). Since x is the furthest point from the supporting line ℓ of s , the reflex chain between x and x' is separated from s by a line. Segment ab' lies in the convex hull of the reflex chain, and so it cannot contain a side of s . $\square$

Proposition 4. *Subroutine EXPAND(t, γ) (i) increases the interior of polygon P , (ii) keeps r as a vertex of P , and (iii) maintains a valid DIRECTED-FOREST convex partition. Furthermore, EXPAND(t, γ) modifies directed edges of the extension tree t only.* $\square$

Lemma 2. *The subroutine FLEXTREE(t) modifies an extension tree $t \in \mathcal{T}$, with a forbidden extended-path γ , in a finite number of EXPAND(t, γ) steps until an obstacle vertex $v' \in V$ appears along γ .*

Proof. FLEXTREE(t) repeatedly calls EXPAND(t, γ) for a forbidden extended-path γ . We associate an integer count(t, γ) to t and γ and show that EXPAND(t, γ) either deforms t to collide with an obstacle $s \neq s'$ or count(t, γ) strictly decreases. This implies that FLEXTREE(t) terminates in a finite number of steps.

Let k denote the size of t (i.e., the number of extended-paths in t). Then t has at most $k - 1$ Steiner vertices in the free space, since each corresponds to the merging of two or more extended-paths. Let k_{ex} be the number of Steiner vertices of t in the exterior of P , let r_P be the number of vertices of P , let f_P be the number of flexible edges of P , and let m_P be the number of directed edges in t that are incident to vertex x of P from the exterior of P . Then let $\text{count}(t, \gamma) = 2k \cdot k_{\text{ex}} + r_P + f_P + 2m_P$. Recall that a Steiner vertex where P has an internal angle of 180° is not a vertex of P . The vertices of P are v , r and Steiner vertices in the interior of the free space where P has a non-straight internal angle, hence $r_p, f_P, m_P < k$.

Consider a sequence of EXPAND(t, γ) steps where t does not collide with an obstacle. Since in Case 4, a vertex $v' \in V$ appears in the relative interior of t , we may assume that only Case 1–3 are applied. Case 1–3 expand the interior of polygon P , and the directed edges in the exterior of P are not deformed. Hence k_{ex} never increases, and it decreases if P expands and reaches a Steiner point in the exterior of P .

Now consider a sequence of $\text{EXPAND}(t, \gamma)$ steps where k_{ex} remains fixed and Case 4 does not apply. Then m_P can only decrease in Case 1–3. Case 2 initially introduces a new edge of P (increasing r_P and f_P by one each) but it also decreases m_P by at least one. Case 1 and 3 never increase r_P or f_P . In Case 1–3, the deformation step terminates when an interior angle of a convex cell within P becomes 180° (and an edge becomes inflexible, decreasing f_P) or an interior angle of P becomes 180° (and P loses a vertex, decreasing r_P). In both events, $r_P + f_P$ decreases by at least one. Therefore, $\text{count}(t, \gamma) = 2k \cdot k_{\text{ex}} + r_P + f_P + 2m_P$ strictly decreases in every step $\text{EXPAND}(t, \gamma)$, until the relative interior of t collides with an obstacle. $\square$

Theorem 2. *For every finite set of disjoint convex polygonal obstacles in the plane, there is a convex partition and an assignment σ such that the dual graph $D(C, \sigma)$ is 2-edge connected. For k convex polygonal obstacles with a total of n vertices, the convex partition consists of $n - k + 1$ convex cells.*

Proof. The convex partitioning algorithm first creates a STRAIGHT-FORWARD convex partition for the given set of disjoint polygonal obstacles. For k disjoint obstacles with a total of n vertices, it consists of $n - k + 1$ convex cells. The extensions in the convex partition can be represented as a set of extension trees $\mathcal{T}$. We showed in Lemma 1 that there is a bridge in the dual graph iff some extension tree contains a forbidden extended-path. Subroutine $\text{FLEXTREE}(t)$ splits every extension tree t containing a forbidden extended-path into two smaller trees. (The extended-paths in t are distributed between the two resulting trees.) An extension tree that consists of a single extended-path is a straight-line extension, and cannot be forbidden (a straight-line extension emitted from a vertex of an obstacle cannot hit the same obstacle, since each obstacle is convex.) Therefore, after at most $|V|/2$ calls to $\text{FLEXTREE}(t)$, no extended-path is forbidden, and so the dual graph of the convex partition is 2-edge connected. $\square$

References

1. Aichholzer, O., Bereg, S., Dumitrescu, A., García, A., Huemer, C., Hurtado, F., Kano, M., Márquez, A., Rappaport, D., Smorodinsky, S., Souvaine, D., Urrutia, J., Wood, D.: Compatible geometric matchings. *Comput. Geom.* 42, 617–626 (2009)
2. Andrzejak, A., Aronov, B., Har-Peled, S., Seidel, R., Welzl, E.: Results on k -sets and j -facets via continuous motion. In: *Proc. 14th SCG*, pp. 192–199. ACM, New York (1998)
3. Banchoff, T.F.: Global geometry of polygons I: The theorem of Fabricius-Bjerre. *Proc. AMS* 45, 237–241 (1974)
4. Benbernou, N., Demaine, E.D., Demaine, M.L., Hoffmann, M., Ishaque, M., Souvaine, D.L., Tóth, C.D.: Disjoint segments have convex partitions with 2-edge connected dual graphs. In: *Proc. CCCG*, pp. 13–16 (2007)
5. Benbernou, N., Demaine, E.D., Demaine, M.L., Hoffmann, M., Ishaque, M., Souvaine, D.L., Tóth, C.D.: Erratum for Disjoint segments have convex partitions with 2-edge connected dual graphs. In: *Proc. CCCG*, p. 223 (2008)

6. Bose, P., Houle, M.E., Toussaint, G.T.: Every set of disjoint line segments admits a binary tree. *Discrete Comput. Geom.* 26(3), 387–410 (2001)
7. Carlsson, J.G., Armbruster, B., Ye, Y.: Finding equitable convex partitions of points in a polygon efficiently. *ACM Transactions on Algorithms* (to appear, 2009)
8. Chazelle, B., Dobkin, D.P.: Optimal Convex Decompositions. *Comput. Geom.* 2, 63–133 (1985)
9. Kaneko, A., Kano, M.: Perfect partitions of convex sets in the plane. *Discrete Comput. Geom.* 28(2), 211–222 (2002)
10. Keil, M.: Polygon decomposition. In: Sack, J.-R., Urrutia, J. (eds.) *Handbook of Computational Geometry*, pp. 491–518. Elsevier, Amsterdam (2000)
11. Keil, M., Snoeyink, J.: On the time bound for convex decomposition of simple polygons. *Internat. J. Comput. Geom. Appl.* 12, 181–192 (2002)
12. Krumme, D.W., Rafalin, E., Souvaine, D.L., Tóth, C.D.: Tight bounds for connecting sites across barriers. *Discrete Comput. Geom.* 40(3), 377–394 (2008)
13. Lien, J.-M., Amato, N.M.: Approximate convex decomposition of polygons. *Comput. Geom.* 35(1-2), 100–123 (2006)
14. Lingas, A.: The power of non-rectilinear holes. In: Nielsen, M., Schmidt, E.M. (eds.) *ICALP 1982. LNCS*, vol. 140, pp. 369–383. Springer, Heidelberg (1982)

The Closest Pair Problem under the Hamming Metric

Kerui Min¹, Ming-Yang Kao², and Hong Zhu³

¹ School of Computer Science, Fudan University, Shanghai, China
krmin@fudan.edu.cn

² Department of Electrical Engineering and Computer Science, Northwestern
University, Evanston, IL, United States
kao@northwestern.edu

³ Software School, East China Normal University, Shanghai, China
hzhu@sei.ecnu.edu.cn

Abstract. Finding the closest pair among a given set of points under Hamming Metric is a fundamental problem with many applications. Let n be the number of points and D the dimensionality of all points. We show that for $0 < D \leq n^{0.294}$, the problem, with the binary alphabet set, can be solved within time complexity $O\left(n^{2+o(1)}\right)$, whereas for $n^{0.294} < D \leq n$, it can be solved within time complexity $O\left(n^{1.843} D^{0.533}\right)$. We also provide an alternative approach not involving algebraic matrix multiplication, which has the time complexity $O\left(n^2 D / \log^2 D\right)$ with small constant, and is effective for practical use. Moreover, for arbitrary large alphabet set, an algorithm with the time complexity $O\left(n^2 \sqrt{D}\right)$ is obtained for $0 < D \leq n^{0.294}$, whereas the time complexity is $O\left(n^{1.921} D^{0.767}\right)$ for $n^{0.294} < D \leq n$. In addition, the algorithms propose in this paper provides a solution to the open problem stated by Kao et al.

1 Introduction

Problem description. Given a set S of n points and an alphabet set Σ . Each point has D dimensions, *i.e.* $p(i) \in \Sigma, i = 1 \cdots D$, where $p(i)$ denotes the i -th component of p . Similarly, $p(i..j)$ denotes the components of p from i to j inclusive. For any $p, q \in S$, define $D_H(p, q)$ as the Hamming distance between p and q , $D_H(p(i), q(j)) = 0$ if and only if $p(i) = q(j)$. The aim of the problem is to find any pair p^*, q^* such that $D_H(p^*, q^*) = \min_{p, q \in S} D_H(p, q)$. We use CP as an abbreviation for this problem in the rest of this paper. And we use CP_{01} to denote the case that the alphabet set $\Sigma = \{0, 1\}$.

Motivation and Related Work. The CP problem has many applications in varying fields, such as information retrieval, data clustering, error-correcting code design, and DNA sequence comparison [2,6,10]. For example, the classic single linkage clustering algorithm treats every point as a separate cluster at its

initial step. For each iteration that follows, the algorithm finds the closest pair and combine them into one cluster. For another example, in error-correcting code design, the distance between the closest pair codewords is an important measurement to indicate directly, the error correction capability of the code.

Several problems are closely related to CP, including nearest neighbor, range search, and bit vector intersection problem [13,14]. Obviously, an efficient nearest neighbor solution could naturally yield a CP algorithm. The nearest neighbor problem was intensively studied for the past decades, and substantial progress has been made [13]. In that paper, instead of tackling the exact answer, P. Indyk and R. Motwani relaxed the exact nearest neighbor to approximated answers. With the technique of Locality-sensitive-hashing(LSH), one can find c -approximation answer under l_2 in $O(Dn^{1/c^2+o(1)})$ with high probability [1]. Following the approach, [6] studies better hash functions to approximate CP problem. For the high dimension case where $D = n$, [12] shows the exact closest pair under l_1 can be found in $O(n^{(\omega+3)/2})$, and $O(n^\omega)$ for l_2 , where $O(n^\omega)$ is the running time of matrix multiplication. The paper provided the first better-than-naive algorithm for CP in high dimensionality. Considering another case whereby the set S is uniformly distributed, [5] gave an algorithm can check whether the minimal Hamming distance of the given set S is less than a given integer k with expected time complexity $O(kn^{1+k/D})$.

The current lowest exponent record $\omega = 2.376$ for matrix multiplication, was discovered and presented by D. Coppersmith and S. Winograd in 1987 [4]. If we limit the matrix to be of boolean type, there are several non-algebraic approaches to square matrix multiplication. The first sub-cubic algorithm with time complexity $O(n^3/\log n)$ was purposed in [16], the result was then improved to $O(n^3/\log^{1.5} n)$ in [9]. A solution with the time complexity $O(n^3/\log^2 n)$ was given in [17] and rediscovered in [8]. The practical algorithm given in this paper employs the grouping idea in [8] and generalizes it to any binary rectangular matrix.

Our Results. Despite the importance of CP problem, previous results, to the best of our knowledge, either worked on approximate solution, or made certain assumptions (*i.e.* dimensionality, distribution of input). In this paper, we show that when $0 < D \leq n^{0.294}$, CP_{01} can be solved within the time complexity $O(n^{2+o(1)})$, whereas for $n^{0.294} < D \leq n$, it can be solved within the time complexity $O(n^{1.843}D^{0.533})$. An alternative approach not involving algebraic matrix multiplication is provided. It has the time complexity $O(n^2D/\log^2 D)$ with small constant, and is effective for practical application. Moreover, for arbitrary large alphabet set, an algorithm with time complexity $O(n^2\sqrt{D})$ is obtained for $0 < D \leq n^{0.294}$, whereas the time complexity is $O(n^{1.921}D^{0.767})$ for $n^{0.294} < D \leq n$. We also show how that the algorithms proposed in this paper provide a solution to the open problem stated in [10].

2 Preliminaries

In this paper, we assume the standard $(\log n)$ -RAM computation model, that is, arithmetic operations on $\log n$ bits can be done in $O(1)$ time. Let $\mathbf{1}_{m \times n}$ denote an m by n matrix with all entries filled with 1.

There is a straightforward method to reduce the CP_{01} problem to matrix multiplication. Notice that the Hamming distance between p and q can be computed as follow:

$$D_H(p, q) = D - \sum_{i=1}^D p(i)q(i) - \sum_{i=1}^D (1 - p(i))(1 - q(i))$$

The set of points S can be grouped in a n by D matrix A , each row of the matrix representing a point in S , where the i -th row is the i -th point in S . The Hamming distance between the i -th point p and the j -th point q could be rewritten as

$$D_H(p, q) = D - (AA^T)_{ij} - ((\mathbf{1}_{n,D} - A)(\mathbf{1}_{n,D} - A)^T)_{ij}$$

The matrices AA^T and $(\mathbf{1}_{n,D} - A)(\mathbf{1}_{n,D} - A)^T$ need to be computed only once. Using the results, the Hamming distance for any pair of words could be computed in $O(1)$ time. Another reduction given in [12] for l_2 using only one matrix multiplication, which also holds for the case of the binary alphabet set.

$$\begin{aligned} D_H(p, q) &= \sum_{k=1}^D |p(k) - q(k)|^2 = \sum_{k=1}^D p(k)^2 + \sum_{k=1}^D q(k)^2 - 2 \sum_{k=1}^D p(k)q(k) \\ &= \sum_{k=1}^D p(k)^2 + \sum_{k=1}^D q(k)^2 - 2 \sum_{k=1}^D A_{ik}A_{jk} \\ &= \sum_{k=1}^D p(k)^2 + \sum_{k=1}^D q(k)^2 - 2(AA^T)_{ij} \end{aligned} \tag{1}$$

Hence, we focus on the problem of matrix multiplication.

3 Rectangular Matrix Multiplication

In this section, we assume $n \geq D$. The case in which $n < D$ could be treated in the similar way. Without loss of generality, we assume n is divisible by D , as it's possible to attach a zero padded matrix.

Definition 1. Define $\omega(r, s, t)$ to be the lowest exponent of the multiplication of an $n^r \times n^s$ matrix by an $n^s \times n^t$ matrix.

The well known result from [4] indicates that $\omega := \omega(1, 1, 1) \leq 2.376$. An upper bound for $\omega(1, s, 1)$ can be derived from square matrix multiplication directly. We rewrite matrix A as

$$A = [M^{(1)}M^{(2)} \dots M^{(n/D)}]^T$$

where $M^{(i)}$ are D by D square matrices. We get

$$\begin{aligned}
 AA^T &= [M^{(1)}M^{(2)} \dots M^{(n/D)}]^T [M^{(1)}M^{(2)} \dots M^{(n/D)}] \\
 &= \begin{pmatrix} M^{(1)}M^{(1)} & M^{(1)}M^{(2)} & \dots & M^{(1)}M^{(n/D)} \\ \vdots & \vdots & \ddots & \vdots \\ M^{(n/D)}M^{(1)} & M^{(n/D)}M^{(2)} & \dots & M^{(n/D)}M^{(n/D)} \end{pmatrix} \quad (2)
 \end{aligned}$$

Thus the rectangular matrix multiplication can be done by partitioned matrices within time $O(n^2 D^{\omega-2})$, which implies the following lemma.

Lemma 1. $\omega(1, s, 1) \leq 2 + s(\omega - 2)$

However, better bounds can be found in [3,18].

Lemma 2. ([3]) Let $\alpha = \sup\{0 \leq s \leq 1 : \omega(1, s, 1) = 2 + o(1)\}$. It can be proven that $\alpha \geq 0.294$.

Lemma 3. ([18])

$$\omega(1, s, 1) \leq \begin{cases} 2 + o(1), & 0 \leq s \leq \alpha, \\ 2 + \frac{\omega - 2}{1 - \alpha}(s - \alpha) + o(1), & \alpha < s \leq 1. \end{cases}$$

Combine the results of Lemma 2 and Lemma 3, we get the following theorem.

Theorem 1. For $0 < D \leq n^{0.294}$, CP_{01} problem can be solved within the time complexity $O(n^{2+o(1)})$, whereas for $n^{0.294} < D \leq n$, it can be solved within the time complexity $O(n^{1.843} D^{0.533})$.

4 The Practical Algorithm

In this section, we reduce the problem of binary matrix multiplication to a graph problem, and propose a practically feasible solution to the graph problem. The solution can be applied to all applications that require the binary matrix multiplication. Binary matrix is defined as a matrix with every entry containing either 0 or 1, but the result of matrix multiplication could contain entries with any non-negative integer. Notice that by replacing all the positive integers in the resultant matrix to 1, the new matrix created correspond to the result of boolean matrix multiplication, studied in [8,17].

Consider that given a binary $n \times D$ matrix A and a binary $D \times m$ matrix B . We construct a three column graph with $n + D + m$ vertices from matrices A, B . For all i and j , the value $A_{ij} = 1$ indicates that there exists an edge from the i -th vertex of the first column to the j -th vertex of the second column, similarly, $B_{ij} = 1$ indicates an edge from the j -th vertex of the second column to the i -th vertex of the third column.

Let P to be the result of matrix multiplication, i.e. $P = AB$, where $P_{ij} = \sum_{k=1}^D A_{ik} B_{jk}$ is equal to the number of distinct path from the i -th vertex of

Algorithm 1. Batch Process Algorithm

Input: $\mathcal{A}, \mathcal{B}$
Output: P
 $P[1 \cdots n, 1 \cdots m] = 0$
for $i = 1$ **to** n/b **do**
 for $j = 1$ **to** m/b **do**
 $Pattern[0 \cdots 2^b - 1, 0 \cdots 2^b - 1] = 0$
 for $k = 1$ **to** D **do**
 $Pattern[\mathcal{A}_{ik}, \mathcal{B}_{jk}] = Pattern[\mathcal{A}_{ik}, \mathcal{B}_{jk}] + 1$
 end for
 for $s = 0$ **to** $2^b - 1$ **do**
 for $t = 0$ **to** $2^b - 1$ **do**
 if $Pattern[s, t] > 0$ **then**
 $BlockDecode(i, j, s, t, Pattern[s, t])$
 end if
 end for
 end for
 end for
end for

the first column to the j -th vertex of the third column. We'll show how to compute P efficiently. Every path counted in P_{ij} will pass through a vertex in the second column. For any particular k -th vertex of second column, elements in the first column of matrix A , $A_{1k}, A_{2k}, \dots, A_{nk}$ can be seen as a binary sequence. The same property applies to matrix B . Consider to encode every consecutive b bits in A and B respectively to obtain an integer number, denote as $\mathcal{A}_{ik}$, where $\mathcal{A}_{ik} = \sum_{s=1}^b 2^{s-1} A_{(i-1)b+s,k}$ for $1 \leq i \leq n/b$, and $\mathcal{B}_{jk}$, where $\mathcal{B}_{jk} = \sum_{s=1}^b 2^{s-1} B_{(j-1)b+s,k}$ for $1 \leq j \leq m/b$.

We call an encoded b bits, $\mathcal{A}_{ik}$ or $\mathcal{B}_{jk}$, a block. There is no information lost from this encoding. So it's easy to see that for a given $\mathcal{A}_{ik}$ and $\mathcal{B}_{jk}$, it's possible to enumerate all the paths from the i -th block of first column to the j -th block of the third column. Simply enumerating and counting all the paths doesn't outperform the naive algorithm. However, observe that the value k is insignificant to the decoding process. That is, if we know that vertex i has an edge to an intermediate vertex (in the second column) and the intermediate vertex has an edge to the vertex j in the third column, there is always a path from i to j . So instead of performing decoding right after each enumeration, it is better to delay the procedure until all the intermediate vertices were enumerated thus the same pattern could be decoded only once. Such idea is formalized by **Algorithm 1**.

The procedure `BlockDecode` enumerates all the possible paths in the given block and cumulate the number in the result matrix P . The pseudo-code can be seen in **Algorithm 2**. The procedure `Binary indicate` the binary representation of the given integer number.

Now we calculate the time complexity of this algorithm. The outer loop iterates $(n/b)(m/b)$ times, each iteration takes $O(D)$ times for `Pattern`

Algorithm 2. BlockDecode Algorithm

Input: $i, j, s, t, Value$
for $x = 1$ **to** b **do**
 for $y = 1$ **to** b **do**
 if $\text{Binary}(s)[x] = 1$ and $\text{Binary}(t)[y] = 1$ **then**
 $P[(i-1)b+x, (j-1)b+y] = P[(i-1)b+x, (j-1)b+y] + Value$
 end if
 end for
end for

construction, and $O(b^2 2^{2b})$ time for BlockDecode. Thus the total time complexity is $O\left(\frac{nm}{b^2}(D + b^2 2^{2b})\right)$. Let $b = \frac{\log D}{3}$. It is easy to see that the time complexity is bounded by $O(nmD/\log^2 D)$. Therefore $P = AB$ can be calculated in $O(nmD/\log^2 D)$ time, which yields the following theorem.

Theorem 2. *There exists an algorithm not involving algebraic matrix multiplication, that solves CP_{01} problem with the time complexity $O(n^2 D/\log^2 D)$.*

5 Non-binary Alphabet Set

There are certain applications requiring the non-binary alphabet set, for instance in bioinformatics, the DNA alphabet set $\Sigma = \{A, C, G, T\}$ consists of four letters. In this section, we propose two algorithms in dealing with alphabet sets of different sizes.

5.1 Small Alphabet Set

We've shown earlier that CP_{01} can be solved efficiently. It's not difficult to find that the non-binary alphabet set could be reduce to CP_{01} through encoding, and be solved by taking the advantage of CP_{01} solutions.

Definition 2. *Denote code $\mathcal{C}(N, M, d)$ to be a binary code of length N , size M and minimum Hamming distance d .*

Definition 3. *$\mathcal{C}(N, M, d)$ is considered an equidistant binary code if all pairwise Hamming distance of codewords in $\mathcal{C}$ are identical.*

We need to construct an equidistant binary code where $M \geq n$, and the length of words N is as short as possible. A straightforward way is to encode the i -th character in Σ to be a length $|\Sigma|$ binary word where only the i -th bit is 1. For example, the four letters DNA alphabet set would be encoded as $\Sigma^* = \{1000, 0100, 0010, 0001\}$, where by any different words has Hamming distance of 2. Notice that $\Sigma^* = \{100, 010, 001, 111\}$ is also a feasible code, and one bit shorter. It is natural to ask what is the shortest length of the equidistant binary code is, while satisfying our requirement. We have the following theorem.

Theorem 3 (Plotkin Bound[7]). *For every (N, M, d) binary code $\mathcal{C}$ for which $N < 2d$, we have*

$$M \leq \frac{2d}{2d - N}$$

Obviously the maximal M can not be greater than $N+1$ from the above theorem. The relationship between N and M of the equidistant binary code is further discussed in [15].

Theorem 4. ([15]) *Let $\mathcal{C}$ be (N, M, d) an equidistant binary code. Then $M \leq N + 1$, and equality holds if and only if $N = 2d - 1$ and $\mathcal{C}$ attains the Plotkin bound.*

Therefore the above encoding is optimal. By applying the algorithm of Theorem 1 and encoding, the following theorem is obtained.

Theorem 5. *For $0 < |\Sigma|D \leq n^{0.294}$, CP problem can be solved within the time complexity $O(n^{2+o(1)})$, whereas for $n^{0.294} < |\Sigma|D \leq n$, it can be solved within the time complexity $O(n^{1.843}(|\Sigma|D)^{0.533})$.*

5.2 Large Alphabet Set

For large alphabet set, i.e. $|\Sigma| = n$, the time complexity of the algorithm above is worse than the naive algorithm. Inspired by the the pattern matching result in [11], we show that there exists an algorithm which the time complexity is independent of the size of alphabet set.

Lemma 4. ([11]) *For any pair of integer x, y*

$$|x - y + 1| + |x - y - 1| - 2|x - y| = \begin{cases} 0, & x \neq y, \\ 2, & x = y. \end{cases}$$

Without loss of generality, assume that $\Sigma = \{1, 2, \dots, |\Sigma|\}$. Notice that the distance between point p and point q under l_1 is equal to $\sum_{i=1}^D |p(i) - q(i)|$. Let the $n \times D$ matrix A be the original matrix consists of n points, $A_{ij} \in \{1, 2, \dots, |\Sigma|\}$. And let A^+ be the matrix where $A_{ij}^+ = A_{ij} + 1$, similarly, $A_{ij}^- = A_{ij} - 1$. The algorithm is shown below.

A two stages algorithm is introduced in [12] to calculate the distance matrix under l_1 . From a high level point of view, their algorithm divide Σ into k groups based on the frequency. In the first stage, the algorithm treats different groups as different characters. The deviation is then corrected in the second stage. The detailed steps refer to [12]. Combine with their analysis and Theorem 1, we have the following theorem by setting the group size $k = \sqrt{D}$ for $D \leq n^{0.294}$, or $k = D^{0.234}n^{0.079}$ for $n^{0.294} < D \leq n$.

Theorem 6. *When $0 < D \leq n^{0.294}$, CP problem with arbitrary large Σ can be solved within the time complexity $O(n^2\sqrt{D})$, whereas for $n^{0.294} < D \leq n$, it can be solved within the time complexity $O(n^{1.921}D^{0.767})$.*

Algorithm 3. $|\Sigma|$ Independent Algorithm**Input:** A **Output:** P Construct the $3n \times D$ matrix $C = \begin{pmatrix} A \\ A^+ \\ A^- \end{pmatrix}$ Calculate the distance matrix E with respect to input matrix C under l_1 **for** $i = 1$ to n **do** **for** $j = 1$ to n **do** $P[i, j] = D - (E[i, i + n] + E[i, i + 2n] - 2E[i, j])/2$ **end for****end for**

6 DNA Words Verification

The problem of designing DNA words was considered by Kao et al [10], where randomized techniques were employed to construct the DNA words with certain combinatorial constraints. Though it has been proven that the given constraints can be satisfied with high probability, there are cases in which the given constraints fail to be satisfied. Finding non-trivial verification algorithms is an open problem which stated in that paper. In this section, as an application of the algorithms proposed earlier in the paper, we provide a solution to the open problem.

First we introduce the notations. Let $p = p(1)p(2) \cdots p(D)$ be a word. Each component $p(i)$ belongs to the alphabet set $\Sigma = \{A, C, G, T\}$. The *reverse* of p , denoted by p^R , is equal to $p(D)p(D-1) \cdots p(1)$. The *complement* $p(i)$ is denoted by $p^C(i)$. Specifically, $A^C=T$, $C^C=G$, $G^C=C$, $T^C=A$. The *complement* of a word is obtained by taking the complement of every component. The *reverse complement* of p , denoted by p^{RC} , is equal to $p^C(D)p^C(D-1) \cdots p^C(1)$. S is the given word set.

There are nine constraints to be verified, which are briefly outlined below. The biological significance of these constraints are omitted.

$C_1(k_1)$: **Basic Hamming Constraint**= $\forall p, q \in S, D_H(p, q) \geq k_1$.

$C_2(k_2)$: **Reverse Complementary Constraint**= $\forall p, q \in S, D_H(p, q^{RC}) \geq k_2$.

$C_3(k_3)$: **Self Complementary Constraint**= $\forall p \in S, D_H(p, p^{RC}) \geq k_3$.

$C_4(k_4)$: **Shifting Hamming Constraint**= $\forall p, q \in S,$

$D_H(p(1..i), q((D-i+1)..D)) \geq k_4 - (D-i)$ for all i .

$C_5(k_5)$: **Shifting Reverse Complementary Constraint**= $\forall p, q \in S,$

$D_H(p(1..i), q^{RC}(1..i)) \geq k_5 - (D-i)$ for all i ;

$D_H(p((D-i+1)..D), q^{RC}((D-i+1)..D)) \geq k_5 - (D-i)$ for all i .

$C_6(k_6)$: **Shifting Self Complementary Constraint**= $\forall p \in S,$

$D_H(p(1..i), p^{RC}(1..i)) \geq k_6 - (D-i)$ for all i ;

$D_H(p((D-i+1)..D), p^{RC}((D-i+1)..D)) \geq k_6 - (D-i)$ for all i .

$C_7(\gamma)$: **GC Content Constraint**= γ percentage of bases in any word $p \in S$ are either G or C.

$C_8(d)$: **Consecutive Base Constraint**= no word has more than d consecutive bases for $d \geq 2$.

$C_9(\sigma)$: **Free Energy Constraint**= $\forall p, q \in S, |FE(p) - FE(q)| \leq \sigma$, where $FE(p)$ denotes the free energy of word p .

Among the nine constraints, C_3, C_6, C_7, C_8, C_9 can be verified trivially in $O(nD)$. Moreover, we have the following observation.

Lemma 5. *Consider the constraint C_4 , for given $p, q \in S$, the inequality $D_H(p(1..i), q((D - i + 1)..D)) \geq k_4 - (D - i)$ hold for all i if and only if $D_H(p, q) \geq k_4$.*

Proof. The sufficiency is obvious. Since the case $i = D$ results $D_H(p, q) \geq k_4$. For necessity, assume that $D_H(p, q) \geq k_4$ but $D_H(p(1..j), q((D - j + 1)..D)) \geq k_4 - (D - j)$ does not hold for j . Thus we have $D_H(p(1..j), q((D - j + 1)..D)) < k_4 - (D - j)$.

$$\begin{aligned} D_H(p(1..(j+1)), q((D-j)..D)) &= D_H(p(1..j), q((D-j+1)..D)) \\ &\quad + D_H(p(j+1), q(D-j)) \\ &< k_4 - (D-j) + 1 \\ &= k_4 - (D-(j+1)) \end{aligned}$$

which implies that $j+1$ does not hold either. Therefore we have $D_H(p, q) < k_4$ by induction, which is contradicts the initial assumption. Thus $D_H(p(1..i), q((D - i + 1)..D)) \geq k_4 - (D - i)$ hold for all i . $\square$

For constraint C_5 , we have the similar result. Let $S^* = \{p : p \in S \vee p^{RC} \in S\}$. By applying the algorithm of Theorem 5 with respect to S^* , the distance matrix under Hamming Metric can be computed efficiently, and then the DNA words verification can be done in $O(n^2)$. The $|S^*| = 2n$ and encoding of the DNA alphabet set only affect the constant of the algorithm.

Theorem 7. *When $0 < D \leq n^{0.294}$, the DNA words verification with constraints C_1 to C_9 can be done within the time complexity $O(n^{2+o(1)})$, whereas for $n^{0.294} < D \leq n$, it can be verified within the time complexity $O(n^{1.843} D^{0.533})$.*

7 Future Work

All algorithms proposed in this paper are devoted to solve the static version of closest pair problem under Hamming Metric. Investigating the dynamic version of CP problem, *i.e.* insertion and deletion are supported, would be both interesting and challenging.

The algorithms proposed in this paper have the running time $\Omega(n^2)$, even for small D , say $D = \sqrt{n}$. Finding algorithms with the complexity $O(n^{2-\epsilon})$ for arbitrary small constant ϵ , or proving the lower bound of CP is $\Omega(n^2)$, is open for future work.

Acknowledgements

Part of this work was done at Hong Kong University. We would like to thank Professor Francis Chin for helpful discussions.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: 47th Annual Symposium on Foundations of Computer Science, pp. 459–468 (2006)
2. Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y.: C2P: clustering based on closest pairs. In: 27th International Conference on Very Large Data Bases, pp. 331–340 (2001)
3. Coppersmith, D.: Rectangular matrix multiplication revisited. *Journal of Complexity* 13(1), 42–49 (1997)
4. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. In: 19th Annual ACM Symposium on Theory of Computing, pp. 1–6 (1987)
5. Greene, D., Parnas, M., Yao, F.: Multi-index hashing for information retrieval. In: 35th Annual Symposium on Foundations of Computer Science, pp. 722–731 (1994)
6. Gordon, D.M., Miller, V., Ostapenko, P.: Optimal hash functions for approximate closest pairs on the n -cube (2008), <http://arxiv.org/abs/0806.3284>
7. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes. North-Holland Mathematical Library (1977)
8. Basch, J., Khanna, S., Motwani, R.: On diameter verification and boolean matrix multiplication. Technical Report No. STAN-CS-95-1544, Department of Computer Science, Stanford University (1995)
9. Atkinson, M.D., Santoro, N.: A practical algorithm for Boolean matrix multiplication. *Information Processing Letters* 29(1), 37–38 (1988)
10. Kao, M.Y., Sanghi, M., Schweller, R.: Randomized fast design of short DNA words. In: 32nd International Colloquium on Automata, Languages, and Programming, pp. 1275–1286 (2005)
11. Lipsky, O., Porat, E.: L1 pattern matching lower bound. *Information Processing Letters* 105(4), 141–143 (2008)
12. Indyk, P., Lewenstein, M., Lipsky, O., Porat, E.: Closest pair problems in very high dimensions. In: 31st International Colloquium on Automata, Languages and Programming, pp. 782–792 (2004)
13. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: 30th Annual Symposium on Theory of Computing, pp. 604–613 (1998)
14. Karp, R.M., Waarts, O., Zweig, G.: The bit vector intersection problem. In: 36th Annual Symposium on Foundations of Computer Science, pp. 621–630 (1995)
15. Roth, R.M., Seroussi, G.: Bounds for binary codes with narrow distance distributions. *IEEE Transactions on Information Theory* 53(8), 2760–2768 (2007)
16. Arlazarov, V.Z., Dinic, E.A., Kronrod, M.A., Faradzev, I.A.: On economical construction of the transitive closure of a directed graph. *Dokl. Akad. Nauk* 194, 487–488 (1970) (in Russian)
17. Rytter, W.: Fast recognition of pushdown automaton and context-free languages. *Information and Control* 67(1-3), 12–22 (1985)
18. Huang, X., Pan, V.Y.: Fast rectangular matrix multiplications and applications. *Journal of Complexity* 14(2), 257–299 (1998)

Space Efficient Multi-dimensional Range Reporting

Marek Karpinski and Yakov Nekrich

Dept. of Computer Science, University of Bonn
`{marek,yasha}@cs.uni-bonn.de`

Abstract. We present a data structure that supports three-dimensional range reporting queries in $O(\log \log U + (\log \log n)^3 + k)$ time and uses $O(n \log^{1+\varepsilon} n)$ space, where U is the size of the universe, k is the number of points in the answer, and ε is an arbitrary constant. This result improves over the data structure of Alstrup, Brodal, and Rauhe (FOCS 2000) that uses $O(n \log^{1+\varepsilon} n)$ space and supports queries in $O(\log n + k)$ time, the data structure of Nekrich (SoCG'07) that uses $O(n \log^3 n)$ space and supports queries in $O(\log \log U + (\log \log n)^2 + k)$ time, and the data structure of Afshani (ESA'08) that uses $O(n \log^3 n)$ space and also supports queries in $O(\log \log U + (\log \log n)^2 + k)$ time but relies on randomization during the preprocessing stage. Our result allows us to significantly reduce the space usage of the fastest previously known static and incremental d -dimensional data structures, $d \geq 3$, at a cost of increasing the query time by a negligible $O(\log \log n)$ factor.

1 Introduction

The range reporting problem is to store a set of d -dimensional points P in a data structure, so that for a query rectangle Q all points in $Q \cap P$ can be reported. In this paper we significantly improve the space usage and pre-processing time of the fastest previously known static and semi-dynamic data structures for orthogonal range reporting with only a negligible increase in the query time.

The range reporting is extensively studied at least since 1970s; the history of this problem is rich with different trade-offs between query time and space usage. Static range reporting queries can be answered in $O(\log^d n + k)$ time and $O(n \log^{d-1} n)$ space using range trees [4] known since 1980; here and further n denotes the number of points in P and k denotes the number of points from P in the query rectangle. The query time can be reduced to $O(\log^{d-1} n + k)$ time by applying the fractional cascading technique of Chazelle and Guibas [8] designed in 1985. The space usage was further improved by Chazelle [6]. In 90s, Subramanian and Ramaswamy [14] and Bozanis, Kitsios, Makris, and Tsakalidis [5] showed that d -dimensional queries can be answered in $\tilde{O}(\log^{d-2} n + k)$ time¹ at a cost of higher space usage: their data structures use $O(n \log^{d-1} n)$ and $O(n \log^d n)$ space

¹ We denote $\tilde{O}(f(n)) = O(f(n) \log^c(f(n)))$ for a constant c .

Table 1. Data structures in $d > 3$ dimensions; † indicates that a data structure is randomized. We define $\log^*(n) = \min\{t \mid \log^{(t)} n \leq 1\}$ and $\log^{**} n = \min\{t \mid \log^{*(t)} n \leq 1\}$ where $\log^{*(t)} n$ denotes computing $\log^*$ t times.

Source	Query Time	Space
[4]	$O(\log^d n + k)$	$O(n \log^{d-1} n)$
[8]	$O(\log^{d-1} n + k)$	$O(n \log^{d-1} n)$
[6]	$O(\log^{d-1} n + k)$	$O(n \log^{d-2+\varepsilon} n)$
[14]	$O(\log^{d-2} n \log^{**} n + k)$	$O(n \log^{d-1} n)$
[5]	$O(\log^{d-2} n + k)$	$O(n \log^d n)$
[2]	$O(\log^{d-2} n / (\log \log n)^{d-3} + k)$	$O(n \log^{d-2+\varepsilon} n)$
[13]	$O(\log^{d-3} n / (\log \log n)^{d-5} + k)$	$O(n \log^{d+1+\varepsilon} n)$
[1]†	$O(\log^{d-3} n / (\log \log n)^{d-5} + k)$	$O(n \log^{d+\varepsilon} n)$
This paper	$O(\log^{d-3} n / (\log \log n)^{d-6} + k)$	$O(n \log^{d-2+\varepsilon} n)$

respectively. Alstrup, Brodal, and Rauhe [2] designed a data structure that answers queries in $\tilde{O}(\log^{d-2} n + k)$ time and uses $O(n \log^{d-2+\varepsilon} n)$ space for an arbitrary constant $\varepsilon > 0$. Nekrich [13] reduced the query time by $\tilde{O}(\log n)$ factor and presented a data structure that answers queries in $O(\log^{d-3} n / (\log \log n)^{d-5} + k)$ time for $d > 3$. Unfortunately, the data structure of [13] uses $O(n \log^{d+1+\varepsilon} n)$ space. Recently, Afshani [1] reduced the space usage to $O(n \log^{d+\varepsilon} n)$; however his data structure uses randomization (during the preprocessing stage). In this paper we present a data structure that matches the space efficiency of [2] at a cost of increasing the query time by a negligible $O(\log \log n)$ factor: our data structure supports queries in $O(\log^{d-3} n / (\log \log n)^{d-6} + k)$ time and uses $O(n \log^{d-2+\varepsilon} n)$ space for $d > 3$. See Table 1 for a more precise comparison of different results.

Our result for d -dimensional range reporting is obtained as a corollary of a three-dimensional data structure that supports queries in $O(\log \log U + (\log \log n)^3 + k)$ time and uses $O(n \log^{1+\varepsilon} n)$ space, where U is the size of the universe, i.e. all point coordinates are positive integers bounded by U . Our three-dimensional data structure is to be compared with the data structure of [2] that also uses $O(n \log^{1+\varepsilon} n)$ space but answers queries in $O(\log n + k)$ time and the data structure of [13] that answers queries in $O(\log \log U + (\log \log n)^3 + k)$ time but needs $O(n \log^4 n)$ space. A more extensive comparison with previous results can be found in the full version of this paper [11]. A corollary of our result is an efficient semi-dynamic data structure that supports three-dimensional queries in $\tilde{O}(\log n + k)$ time and insertions in $O(\log^5 n)$ time. Thus we improve the space usage and the update time of fastest previously known semi-dynamic data structure [13] that supports insertions in $O(\log^8 n)$ time.

If we are ready to pay penalties for each point in the answer, the space usage can be further reduced: we describe a data structure that uses $O(n \log^{d-2} n (\log \log n)^3)$ space and answers queries in $O(\log^{d-3} n (\log \log n)^3 + k \log \log n)$ time. We can also use this data structure to answer emptiness queries (to determine whether query rectangle Q contains points from P) and

one-reporting queries (i.e. to report an *arbitrary* point from $P \cap Q$ if $P \cap Q \neq \emptyset$). This is an $\tilde{O}(\log n)$ factor improvement in query time over the data structure of Alstrup et. al. [2]. Other similar data structures are either slower or require higher penalties for each point in the answer.

Throughout this paper, ε denotes an arbitrarily small constant, and U denotes the size of the universe. If each point in the answer can be output in constant time, we will sometimes say that the query time is $O(f(n))$ (instead of $O(f(n) + k)$). We let $[a, b]$ denote the set of integers $\{i | a \leq i \leq b\}$; The intervals $[a, b)$ and $(a, b]$ denote the same set as $[a, b]$ but without a (resp. without b). We denote by $[b]$ the set $[1, b]$.

In section 3 we describe a space efficient data structure for three-dimensional range reporting on a $[U] \times [U] \times [U]$ grid, i.e. in the case when all point coordinates belong to $[U]$. We also describe a variant of our data structure that uses less space but needs $O(\log \log n)$ time to output each point in the answer. All results of this paper are valid in the word RAM computation model.

2 Preliminaries

We use the same notation as in [15] to denote the special cases of three-dimensional range reporting queries: a product of three half-open intervals will be called a (1,1,1)-sided query; a product of a closed interval and two half-open intervals will be called a (2,1,1)-sided query; a product of two closed intervals and one half-open interval (resp. three closed intervals) will be called a (2,2,1)-sided (resp. (2,2,2)-sided) query. Clearly (1,1,1)-sided queries are equivalent to dominance reporting queries, and (2,2,2)-sided query is the general three-dimensional query. The following transformation is described in e.g. [15] and [14].

Lemma 1. *Let $1 \leq a_i \leq b_i \leq 2$ for $i = 1, 2, 3$. A data structure that answers (a_1, a_2, a_3) queries in $O(q(n))$ time, uses $O(s(n))$ space, and can be constructed in $O(c(n))$ time can be transformed into a data structure that answers (b_1, b_2, b_3) queries in $O(q(n))$ time, uses $O(s(n) \log^t n)$ space and can be constructed in $O(c(n) \log^t n)$ time for $t = (b_1 - a_1) + (b_2 - a_2) + (b_3 - a_3)$.*

We say that a set P is on a grid of size n if all coordinates of all points in P belong to an interval $[n]$. We will need the following folklore result:

Lemma 2. *There exists a $O(n^{1+\varepsilon})$ space data structure that supports range reporting queries on a d -dimensional grid of size n for any constant d in $O(k)$ time.*

Proof. One dimensional range reporting queries on the $[n] \times [n] \times [n]$ grid can be answered in $O(k)$ time using a trie with node degree n^ε . Using range trees [4] with node degree ρ we can transform a d -dimensional $O(s(n))$ space data structure into a $(d + 1)$ -dimensional data structure that uses $O(s(n)h(n) \cdot \rho)$ space and answers range reporting queries in $O(q(n)h(n))$ time, where $h(n) = \log n / \log \rho$ is the height of the range tree. Since $\rho = n^\varepsilon$, $h(n) = O(1)$. Hence, the query time does not depend on dimension and the space usage increases by a factor $O(n^\varepsilon)$ with each dimension. $\square$

We use Lemma 2 to obtain a data structure that supports queries that are a product of a $(d-1)$ -dimensional query on a universe of size $n^{1-\varepsilon}$ and a half-open interval. We will show in the next Lemma that such queries can be answered in $O(n)$ space and $O(1)$ time.

Lemma 3. *There exists a $O(n)$ space data structure that supports range reporting queries of the form $Q' \times [-\infty, x)$ in $O(k)$ time, where Q' is a $(d-1)$ -dimensional query on $[U_1] \times [U_2] \times \dots \times [U_{d-1}]$ and $U_1 \cdot U_2 \cdot \dots \cdot U_{d-1} = O(n^{1-\varepsilon})$.*

Proof. There are $O(n^{1-\varepsilon})$ possible projections of points onto the first $d-1$ coordinates. Let $\min(p_1, \dots, p_{d-1})$ denote the point with minimal d -th coordinate among all points whose first $d-1$ coordinates equal to $p_1, p_2, \dots, p_{d-1}$. We store points $\min(p_1, \dots, p_{d-1})$ for all $p_1 \in [U_1], p_2 \in [U_2], \dots, p_{d-1} \in [U_{d-1}]$ in a data structure M . Since M contains $O(n^{1-\varepsilon})$ points, we can use Lemma 2 and implement M in $O(n)$ space. For all possible $p_1 \in [U_1], p_2 \in [U_2], \dots, p_{d-1} \in [U_{d-1}]$ we also store a list $L(p_1, \dots, p_{d-1})$ of points whose first $d-1$ coordinates are $p_1, \dots, p_{d-1}$; points in $L(p_1, \dots, p_{d-1})$ are sorted by their d -th coordinates. Given a query $Q = Q' \times [-\infty, x)$, we first answer Q using the data structure M . Since M contains $O(n^{1-\varepsilon})$ points, we can find all points in $M \cap Q$ in $O(|M \cap Q|)$ time. Then, for every point $p = (p_1, \dots, p_{d-1}, p_d)$ found with help of M , we traverse the corresponding list $L(p_1, \dots, p_{d-1})$ and report all points in this list whose last coordinate does not exceed x . $\square$

In several places of our proofs we will use the *reduction to rank space* technique [10,6]. This technique allows us to replace coordinates of a point by its rank. Let P_x, P_y , and P_z be the sets of x -, y -, and z -coordinates of points from P . For a point $p = (p_x, p_y, p_z)$, let $p' = (\text{rank}(p_x, P_x), \text{rank}(p_y, P_y), \text{rank}(p_z, P_z))$, where $\text{rank}(e, S)$ is defined as the number of elements in S that are smaller than or equal to e . A point p belongs to an interval $[a, b] \times [c, d] \times [e, f]$ if and only if a point p' belongs to an interval $[a', b'] \times [c', d'] \times [e', f']$ where $a' = \text{succ}(a, P_x)$, $b' = \text{pred}(b, P_x)$, $c' = \text{succ}(c, P_y)$, $d' = \text{pred}(d, P_y)$, $e' = \text{succ}(e, P_z)$, $f' = \text{pred}(f, P_z)$, and $\text{succ}(e, S)$ ($\text{pred}(e, S)$) denotes the smallest (largest) element in S that is greater (smaller) than or equal to e . Reduction to rank space can be used to reduce range reporting queries to range reporting on the $[n] \times [n] \times [n]$ grid: Suppose we can find $\text{pred}(e, S)$ and $\text{succ}(e, S)$ for any e , where S is P_x, P_y , or P_z , in time $f(n)$. Suppose that range reporting queries on $[n] \times [n] \times [n]$ grid can be answered in time $O(g(n) + k)$. Then we can answer range reporting queries in $O(f(n) + g(n) + k)$ time. Following [2], we can also use the reduction to rank space technique to reduce the space usage: if a data structure contains m elements, reduction to rank space allows us to store each element in $O(\log m)$ bits.

3 Space Efficient Three-Dimensional Data Structure

In this section we describe a data structure that supports three-dimensional range reporting queries in $O((\log \log n)^3 + \log \log U + k)$ time where U is the universe size and uses $O(n \log^{1+\varepsilon} n)$ space. Our data structure combines the

recursive divide-and-conquer approach introduced in [2], the result of Lemma 3, and the transformation of (a_1, a_2, a_3) -queries into (b_1, b_2, b_3) -queries described in Lemma 1. We start with a description of a space efficient modification of the data structure for $(1,1,1)$ -sided queries on the $[n] \times [n] \times [n]$ grid. Then, we obtain data structures for $(2, 1, 1)$ -sided and $(2, 2, 1)$ -sided queries on the $[n] \times [n] \times [n]$ grid using the recursive divide-and-conquer and Lemma 3. Finally, we obtain the data structure that supports arbitrary orthogonal queries on the $[n] \times [n] \times [n]$ grid using Lemma 1. Reduction to rank space technique described in section 2 allows us to transform a data structure on the $[n] \times [n] \times [n]$ grid into a data structure on the $[U] \times [U] \times [U]$ grid, so that the query time increases by an additive term $O(\log \log U)$ and the space usage is not increased.

Lemma 4. [13] *Given a set of three-dimensional points P and a parameter t , we can construct in $O(n \log^3 n)$ time a $O(n)$ space data structure T that supports the following queries on a grid of size n :*

- (i) *for a given query point q , T determines in $O((\log \log n)^2)$ time whether q is dominated by at most t points of P*
- (ii) *if q is dominated by at most t points from P , T outputs in $O(t + (\log \log n)^2)$ time a list L of $O(t)$ points such that L contains all points of P that dominate q .*

As described in [13], Lemma 4 allows us to answer $(1,1,1)$ -sided queries in $O((\log \log n)^2)$ time and $O(n \log n)$ space. We can reduce the space usage to $O(n \log \log n)$ using an idea that is also used in [1].

Lemma 5. *There exists a data structure that answers $(1,1,1)$ -sided queries on $[n] \times [n] \times [n]$ grid in $O((\log \log n)^2 + k)$ time, uses $O(n \log \log n)$ space, and can be constructed in $O(n \log^3 n \log \log n)$ time.*

Proof. For each parameter $t = 2^{2^i}$, $i = i_{\min}, i_{\min}+1, \dots, \log \log n/2$, $i_{\min} = 2 \log \log \log n$, we construct a data structure T_i of Lemma 4. Given a query point q , we examine data structures T_i , $i = i_{\min}, i_{\min}+1, \dots, \log \log n/2$ until q is dominated by at most 2^{2^i} points of P or the last data structure T_i is examined. Thus we identify the index l , such that q is dominated by more than 2^{2^l} and less than $2^{2^{l+2}}$ points or determine that q is dominated by at least $\log n$ points. If $l = i_{\min}$, then q is dominated by $O((\log \log n)^2)$ points. We can generate in $O((\log \log n)^2)$ time a list L of $O((\log \log n)^2)$ points that contains all points dominating q . Then, we examine all points in L and output all points that dominate q in $O((\log \log n)^2)$ time. If $\log \log n/2 > l > i_{\min}$, we can examine data structures $T_{i_{\min}}, T_{i_{\min}+1}, \dots, T_l$ in $O((l - i_{\min})(\log \log n)^2)$ time. Then, we generate the list L that contains all points that dominate q in $O(2^{2^l})$ time. We can process L and output all k points that dominate q in $O(2^{2^l})$ time. Since $k > 2^{2^{l-2}}$, $k = \Omega(2^{2^l})$ and $k = \Omega((l - i_{\min}) \cdot (\log \log n)^2)$. Hence, the query is answered in $O(k)$ time. If $l = \log \log n/2$, then q is dominated by $\Omega(\log n)$ points. in this case we can use a linear space data structure with $O(\log n)$ query time, e.g. the data structure of Chazelle and Edelsbrunner [7], to answer the query in $O(\log n + k) = O(k)$ time.

Since each data structure T_i uses linear space, the space usage of the described data structure is $O(n \log \log n)$. □

Lemma 6. *There exists a data structure that answers $(2,1,1)$ -sided queries on $[n] \times [n] \times [n]$ grid in $O((\log \log n)^3 + k)$ time, uses $O(n \log^\varepsilon n)$ space, and can be constructed in $O(n \log^3 n \log \log n)$ time.*

Proof. We divide the grid into x -slices $X_i = [x_{i-1}, x_i] \times [n] \times [n]$ and y -slices $Y_j = [n] \times [y_{j-1}, y_j] \times [n]$, so that each x -slice contains $n^{1/2+\gamma}$ points and each y -slice contains $n^{1/2+\gamma}$ points; the value of a constant γ will be specified below. The cell C_{ij} is the intersection of the i -th x -slice and the j -th y -slice, $C_{ij} = X_i \cap Y_j$. The data structure D_t contains a point (i, j, z) for each point $(x, y, z) \in P \cap C_{ij}$. Since the first two coordinates of points in D_t are bounded by $n^{1/2-\gamma}$, D_t uses $O(n)$ space and supports $(2,1,1)$ -sided queries in constant time by Lemma 3. For each x -slice X_i there are two data structures that support two types of $(1,1,1)$ -sided queries, open in $+x$ and in $-x$ directions. For each y -slice Y_j , there is a data structure that supports $(1,1,1)$ -sided queries open in $+y$ direction. For each y -slice Y_j and for each x -slice X_i there are recursively defined data structures. Recursive subdivision stops when the number of elements in a data structure is smaller than a predefined constant. Hence, the number of recursion levels is $v \log \log n$ for $v = \log_{\frac{2}{1+2\gamma}} 2$.

Essentially we apply the idea of [2] to three-dimensional $(2,1,1)$ -sided queries. If a query spans more than one x -slab and more than one y -slab, then it can be answered by answering two $(1,1,1)$ -sided queries, one special $(2,1,1)$ -sided query that can be processed using the technique of Lemma 3, and one $(2,1,1)$ -sided query to a data structure with $n^{1/2+\gamma}$ points. If a query is contained in a slab, then it can be answered by a data structure that contains $n^{1/2+\gamma}$ points. We will show below that the query time is $O((\log \log n)^3)$. Each point is stored in $O(2^i)$ data structures on recursion level i , but space usage can be reduced because the number of points in data structures quickly decreases with the recursion level. We will show below that every point in a data structure on recursion level i can be stored with approximately $(\log n/2^i) \log^{\varepsilon'} n$ bits for an arbitrarily small ε' .

Query Time. Given a query $Q = [a, b] \times (-\infty, c] \times (-\infty, d]$ we identify the indices i_1 , i_2 , and j_1 such that projections of all cells C_{ij} , $i_1 < i < i_2$, $j < j_1$, are entirely contained in $[a, b] \times (-\infty, c]$. Let $a_0 = x_{i_1}$, $b_0 = x_{i_2-1}$, and $c_0 = y_{j_1-1}$. The query Q can be represented as $Q = Q_1 \cup Q_2 \cup Q_3 \cup Q_4$, where $Q_1 = [a_0, b_0] \times (-\infty, c_0] \times (-\infty, d]$, $Q_2 = [a, a_0] \times (-\infty, c] \times (-\infty, d]$, $Q_3 = (b_0, b] \times (-\infty, c] \times (-\infty, d]$, and $Q_4 = [a_0, b_0] \times (c_0, c] \times (-\infty, d]$. Query Q_1 can be answered using D_t . Queries Q_2 and Q_3 can be represented as $Q_2 = ([-\infty, a_0] \times (-\infty, c] \times (-\infty, d]) \cap X_{i_1}$ and $Q_3 = ((-\infty, b] \times (-\infty, c] \times (-\infty, d]) \cap X_{i_2}$; hence, Q_2 and Q_3 are equivalent to $(1,1,1)$ -sided queries on x -slices X_{i_1} and X_{i_2} . The query Q_4 can be answered by a recursively defined data structure for the y -slice Y_{j_1} because $Q_4 = ([a_0, b_0] \times (-\infty, c] \times (-\infty, d]) \cap Y_{j_1}$. If $i_1 = i_2$ and the query Q is contained in one x -slice, then Q is processed by a recursively defined data structure for the corresponding x -slice. Thus a query is reduced to one special case that can be processed in constant time, two $(1,1,1)$ -sided queries, and one $(2,1,1)$ -sided query answered by a data structure that contains $n^{1/2+\gamma}$ elements.

Queries Q_2 and Q_3 can be answered in $O((\log \log n)^2)$ time, the query Q_1 can be answered in constant time. The query Q_4 is answered by a recursively defined data structure that contains $O(n^{1/2+\gamma})$ elements. If $i_1 = i_2$ or $j_1 = 1$, i.e. if Q is entirely contained in one x -slice or one y -slice, then the query is answered by a data structure for the corresponding slice that contains $O(n^{1/2+\gamma})$ elements. Hence, the query time $q(n) = O((\log \log n)^2) + q(n^{1/2+\gamma})$ and $q(n) = O((\log \log n)^3)$.

Space Usage. The data structure consists of $O(\log \log n)$ recursion levels. The total number of points in all data structures on the i -th recursion level is $2^i n$. Hence all data structures on the i -th recursion level require $O(2^i n \log n)$ bits of space. The space usage can be reduced by applying the reduction to rank space technique [10,6]. As explained in section 2, reduction to rank space allows us to replace point coordinates by their ranks. Hence, if we use this technique with a data structure that contains m elements, each point can be specified with $O(\log m)$ bits. Thus, we can reduce the space usage by replacing point coordinates by their ranks on certain recursion levels.

We apply reduction to rank space on every $\delta \log \log n$ -th recursion level for $\delta = \varepsilon/3$. Let V be an arbitrary data structure on recursion level $r = s\delta \log \log n - 1$ for $1 \leq s \leq (1/\delta) \log_{\frac{2}{1+2\gamma}} 2$. Let W be the set of points that belong to an x -slice or a y -slice of V . We store a dictionary that enables us to find for each point $p = (p_x, p_y, p_z)$ from W a point $p' = (p'_x, p'_y, p'_z)$ where $p'_x = \text{rank}(p_x, W_x)$, $p'_y = \text{rank}(p_y, W_y)$, $p'_z = \text{rank}(p_z, W_z)$, and W_x, W_y , and W_z are the sets of x -, y -, and z -coordinates of all points in W . Let W' be the set of all points p' . Conversely there is also a dictionary that enables us to find for a point $p' \in W'$ the corresponding $p \in W$. The data structure that answers queries on W stores points in the rank space of W . In general, all data structures on recursion levels $r, r+1, \dots, r+\delta \log \log n - 1$ obtained by subdivision of W store points in rank space of W . That is, point coordinates in all those data structures are integers bounded by $|W|$. If such a data structure R is used to answer a query Q , then for each point $p_R \in R \cap Q$, we must find the corresponding point $p \in P$. Since range reduction was applied $O(1)$ time, we can find for any $p_R \in R$ the corresponding $p \in P$ in $O(1)$ time.

Each data structure on level $r = s\delta \log \log n$ for $0 \leq s \leq (1/\delta)v$ and $v = \frac{1}{\log(2/(1+2\gamma))}$ contains $O(n^l)$ elements for $l = (1/2 + \gamma)^r$. Hence an arbitrary element of a data structure on level r can be specified with $l \cdot \log n$ bits. The total number of elements in all data structures on the r -th level is $n2^r$. Hence all elements in all data structures on the r -th recursion level need $O(n2^r((\frac{1+2\gamma}{2})^r) \log n \log \log n)$ bits.

We choose γ so that $(1+2\gamma) \leq 2^{\delta/2}$. Then $v = \frac{1}{1-\log_2(1+2\gamma)} \geq \frac{1}{1-\delta/2}$ and $(1+2\gamma) \leq 2^{\delta/2} \leq 2^{\delta-\delta^2/2} \leq 2^{\delta/v} = 2^{\varepsilon/3v}$. Since $r \leq v \log \log n$, $(1+2\gamma)^r \leq 2^{(\varepsilon/3) \log \log n} \leq \log^{\varepsilon/3} n$. Therefore all data structures on level r use $\log^{\varepsilon/3} n \cdot O(n \log n \log \log n) = O(n \log^{1+2\varepsilon/3} n)$ bits of space or $O(n \log^{2\varepsilon/3} n)$ words of $\log n$ bits. The number of elements in all data structures on levels $r+1, r+2, \dots$ increases by a factor two in each level. Hence, the total space (measured in

words) needed for all data structures on all levels q , $r \leq q < r + \delta \log \log n$, is $(\sum_{f=1}^{\delta \log \log n-1} 2^f) O(n \log^{2\varepsilon/3} n) = O(n 2^{\delta \log \log n} n \log^{2\varepsilon/3} n) = O(n \log^\varepsilon n)$ because $\delta \leq \varepsilon/3$ and $2^{\delta \log \log n} \leq \log^{\varepsilon/3} n$. Thus all data structures in a group of $\delta \log \log n$ consecutive recursion levels use $O(n \log^\varepsilon n)$ words of space. Since there are $(1/\delta)v = O(1)$ such groups of levels, the total space usage is $O(n \log^\varepsilon n)$.

Construction Time. The data structure on level 0 (the topmost recursion level) can be constructed in $O(n \log^3 n \log \log n)$ time. The total number of elements in all data structures on level s is $2^s n \log \log n$. But each data structure on the r -th recursion level contains at most $n_r = n^l$ elements and can be constructed in $O(l^3 \cdot n_r \log^3 n \log \log n)$ time where $l = (1 + 2\gamma)^r / 2^r$. Hence, all data structure on the r -th recursion level can be constructed in $O((2^r l^3) n \log^3 n \log \log n) = O(((1 + 2\gamma)^{3r} / 2^{2r}) n \log^3 n \log \log n)$ time. We can assume that $\varepsilon < 1$. Since we chose γ so that $(1 + 2\gamma) \leq 2^{\varepsilon/6}$, $(1 + 2\gamma)^3 < 2$; hence, $(1 + 2\gamma)^{3r} / 2^{2r} \leq 1/2^r$. Then, all data structure on the r -th recursion level can be constructed in $O((1/2^r) n \log^3 n \log \log n)$ time. Summing up by all r , we see that all recursive data structures can be constructed in $O(n \log^3 n \log \log n)$ time. $\square$

Lemma 7. *There exists a data structure that answers $(2,2,1)$ -sided queries on $[n] \times [n] \times [n]$ grid in $O((\log \log n)^3 + k)$ time, uses $O(n \log^\varepsilon n)$ space, and can be constructed in $O(n \log^3 n \log \log n)$ time.*

Proof. The proof technique is the same as in Lemma 6. The grid is divided into x -slices $X_i = [x_{i-1}, x_i] \times n \times n$ and y -slices $Y_j = n \times [y_{j-1}, y_j] \times n$ in the same way as in the proof of Lemma 6. Each x -slice X_i supports $(2, 1, 1)$ -sided queries open in $+x$ and $-x$ direction; each y -slice Y_j supports $(2, 1, 1)$ -sided queries open in $+y$ and $-y$ direction. All points are also stored in a data structure D_t that contains a point (i, j, z) for each point $(x, y, z) \in P \cap C_{ij}$. For every x -slice and y -slice there is a recursively defined data structure. The reduction to rank space technique is applied on every $\delta \log \log n$ -th level in the same way as in the Lemma 6.

Given a query $Q = [a, b] \times [c, d] \times (-\infty, e]$ we identify indices i_1, i_2, j_1, j_2 such that all cells C_{ij} , $i_1 < i < i_2$ and $j_1 < j < j_2$ are entirely contained in Q . Then Q can be represented as a union of a query $Q_1 = [a_0, b_0] \times [c_0, d_0] \times (-\infty, e]$ and four $(2, 1, 1)$ -sided queries $Q_2 = [a, a_0] \times [c, d] \times (-\infty, e]$, $Q_3 = [b_0, b] \times [c, d] \times (-\infty, e]$, $Q_4 = [a_0, b_0] \times [c, c_0] \times (-\infty, e]$, and $Q_5 = [a_0, b_0] \times [d_0, d] \times (-\infty, e]$, where $a_0 = x_{i_1}$, $b_0 = x_{i_2-1}$, $c_0 = y_{j_1}$, and $d_0 = y_{j_2-1}$. The query Q_1 can be answered in constant time, and queries Q_i , $1 < i \leq 5$, can be answered using the corresponding x - and y -slices. Since queries Q_i , $1 < i \leq 5$, are equivalent to $(2,1,1)$ -sided queries each of those queries can be answered in $O((\log \log n)^3 + k)$ time.

If the query Q is entirely contained in one x -slice or one y -slice, then Q is processed by a data structure for the corresponding x -slice resp. y -slice. Since the data structure consists of at most $v \log \log n$ recursion levels, the query can be transferred to a data structure for an x - or y -slice at most $v \log \log n$ times

for $v = \frac{1}{\log(2/(1+2\gamma))}$. Hence, the total query time is $O(\log \log n + (\log \log n)^3 + k) = O((\log \log n)^3 + k)$. The space usage and construction time are estimated in the same way as in Lemma 6. $\square$

Theorem 1. *There exists a data structure that answers three-dimensional orthogonal range reporting queries on the $[U] \times [U] \times [U]$ grid in $O(\log \log U + (\log \log n)^3 + k)$ time, uses $O(n \log^{1+\varepsilon} n)$ space, and can be constructed in $O(n \log^4 n \log \log n)$ time.*

Proof. The result for the $[n] \times [n] \times [n]$ grid directly follows from Lemma 7 and Lemma 1. We can obtain the result for the $[U] \times [U] \times [U]$ grid by applying the reduction to rank space technique [10,6]: We can use the van Emde Boas data structure [9] to find $\text{pred}(e, S)$ and $\text{succ}(e, S)$ for any $e \in [U]$ in $O(\log \log U)$ time, where $S \subset [U]$ is P_x , P_y , or P_z . Hence, the query time is increased by an additive term $O(\log \log U)$ and the space usage remains unchanged. $\square$

Furthermore, we also obtain the result for d -dimensional range reporting, $d \geq 3$.

Corollary 1. *There exists a data structure that answers d -dimensional orthogonal range reporting queries in $O(\log^{d-3} n / (\log \log n)^{d-6} + k)$ time, uses $O(n \log^{d-2+\varepsilon} n)$ space, and can be constructed in $O(n \log^{d+1+\varepsilon} n)$ time.*

Proof. We can obtain a d -dimensional data structure from a $(d-1)$ -dimensional data structure using range trees with node degree $\log^\varepsilon n$. See e.g. [2], [13] for details. $\square$

Using Theorem 1 we can reduce the space usage and update time of the semi-dynamic data structure for three-dimensional range reporting queries.

Corollary 2. *There exists a data structure that uses $O(n \log^{1+\varepsilon} n)$ space, and supports three-dimensional orthogonal range reporting queries in $O(\log n (\log \log n)^2 + k)$ time and insertions in $O(\log^{5+\varepsilon} n)$ time.*

Proof. We can obtain the semi-dynamic data structure from the static data structure using a variant of the logarithmic method [3]. A detailed description can be found in [13]. The space usage remains the same, the query time increases by a $O(\log n / \log \log n)$ factor, and the amortized insertion time is $O(\frac{c(n)}{n} \log^{1+\varepsilon} n)$, where $c(n)$ is the construction time of the static data structure. $\square$

The result of Corollary 2 can be also extended to $d > 3$ dimensions using range trees.

We can further reduce the space usage of the three-dimensional data structure if we allow $O(\log \log n)$ penalties for each point in the answer. Such a data structure can also be used to answer emptiness and one-reporting queries. As in the previous section, we design space efficient data structures for $(2, 1, 1)$ -sided and $(2, 2, 1)$ -sided queries. The proof is quite similar to the data structure of section 3 but some parameters must be chosen in a slightly different way.

Theorem 2. *There exists a data structure that answers three-dimensional orthogonal range reporting queries on the $[U] \times [U] \times [U]$ grid in $O(\log \log U + (\log \log n)^3 + k \log \log n)$ time, uses $O(n \log n (\log \log n)^3)$ space, and can be constructed in time $O(n \log^4 n \log \log n)$.*

For completeness, we provide the proof of Theorem 2 in the full version of this paper [11]. Using the standard range trees and reduction to rank space techniques we can obtain a d -dimensional data structure for $d > 3$.

Corollary 3. *There exists a data structure that answers d -dimensional orthogonal range reporting queries for $d > 3$ in $O(\log^{d-3} n (\log \log n)^3 + k \log \log n)$ time, uses $O(n \log^{d-2} n (\log \log n)^3)$ space, and can be constructed in $O(n \log^{d+1} n \log \log n)$ time.*

References

1. Afshani, P.: On Dominance Reporting in 3D. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 41–51. Springer, Heidelberg (2008)
2. Alstrup, S., Brodal, G.S., Rauhe, T.: New Data Structures for Orthogonal Range Searching. In: Proc. FOCS 2000, pp. 198–207 (2000)
3. Bentley, J.L.: Decomposable Searching Problems. Information Processing Letters 8(5), 244–251 (1979)
4. Bentley, J.L.: Multidimensional Divide-and-Conquer. Commun. ACM 23, 214–229 (1980)
5. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New Results on Intersection Query Problems. The Computer Journal 40(1), 22–29 (1997)
6. Chazelle, B.: A Functional Approach to Data Structures and its Use in Multidimensional Searching. SIAM J. on Computing 17, 427–462 (1988); see also FOCS 1985
7. Chazelle, B., Edelsbrunner, H.: Linear Space Data Structures for Two Types of Range Search. Discrete & Computational Geometry 2, 113–126 (1987)
8. Chazelle, B., Guibas, L.J.: Fractional Cascading: I. A Data Structuring Technique. Algorithmica 1, 133–162 (1986); see also ICALP 1985
9. van Emde Boas, P.: Preserving Order in a Forest in Less Than Logarithmic Time and Linear Space. Inf. Process. Lett. 6(3), 80–82 (1977)
10. Gabow, H., Bentley, J.L., Tarjan, R.E.: Scaling and Related Techniques for Geometry Problems. In: Proc. STOC 1984, pp. 135–143 (1984)
11. Karpinski, M., Nekrich, Y.: Space Efficient Multi-Dimensional Range Reporting, arXiv0806.4361
12. Nekrich, Y.: Space Efficient Dynamic Orthogonal Range Reporting. Algorithmica 49(2), 94–108 (2007)
13. Nekrich, Y.: A Data Structure for Multi-Dimensional Range Reporting. In: Proc. SoCG 2007, pp. 344–353 (2007)
14. Subramanian, S., Ramaswamy, S.: The P-range Tree: A New Data Structure for Range Searching in Secondary Memory. In: Proc. SODA 1995, pp. 378–387 (1995)
15. Vengroff, D.E., Vitter, J.S.: Efficient 3-D Range Searching in External Memory. In: Proc. STOC 1996, pp. 192–201 (1996)

Approximation Algorithms for a Network Design Problem

Binay Bhattacharya*, Yuzhuang Hu, and Qiaosheng Shi

School of Computing Science, Simon Fraser University, Burnaby, Canada, V5A 1S6
{binay,yhu1,qshi1}@cs.sfu.ca

Abstract. A class of network design problems, including the k -path/tree/cycle covering problems and some location-routing problems, can be modeled by downwards monotone functions [5]. We consider a class of network design problems, called the p -constrained path/tree/cycle covering problems, obtained by introducing an additional constraint to these problems; i.e., we require that the number of connected components in the optimal solution be at most p for some integer p . The p -constrained path/tree/cycle covering problems cannot be modeled by downwards monotone functions. In this paper, we present a different analysis for the performance guarantee of the algorithm in [5]. As a result of the analysis, we are able to tackle p -constrained path/tree/cycle covering problems, and show the performance bounds of 2 and 4 for p -constrained tree/cycle problems and p -constrained path covering problems respectively.

1 Introduction

Given an undirected graph $G = (V, E)$ with non-negative edge costs and a function $f : 2^V \rightarrow \{0, 1\}$, we consider the network design problem formulated as the following integer program:

$$\begin{aligned} (IP) \quad & \text{Min} \quad \sum_{e \in E} c_e x_e \\ & \text{subject to:} \\ & \sum_{e \in \delta(S)} x_e \geq f(S) & \emptyset \neq S \subset V \\ & x_e \in \{0, 1\} & e \in E. \end{aligned}$$

Here $\delta(S)$ denotes the cross edges between S and $V - S$, c_e represents the cost of edge e , and x_e indicates whether the edge e is included in the solution.

A downwards monotone function f has the following properties: 1) $f(V) = 0$; 2) $f(A) \geq f(B)$, if and only if $A \subseteq B \subseteq V$. In this paper we consider a class of network design problems which can be modeled as an integer program of the type (IP) with downwards monotone functions. A simple example of such problems is the minimum spanning tree problem, where the corresponding downwards monotone function f can be defined as $f(S) = 1$ if and only if $S \subset V$ and 0

* Research was partially supported by MITACS and NSERC.

otherwise. Other examples include the location-routing problem and the k -cycle covering problem. We present below further details on the last two problems.

Location-routing problem: In this problem [7] we are given an undirected graph $G = (V, E)$ and $D \subseteq V$ which denotes a set of depots. We are assuming here that the graph edge costs satisfy the triangle inequality. A non-negative cost, called the opening cost, is associated to each depot in D . We need to select a subset of depots from D and find a cycle cover of the vertices of G (a set of disjoint simple cycles that cover all the vertices in V). Each cycle in the cover must contain a selected depot. The goal is to minimize the cost of the cycle edges plus the opening cost of the selected depots. Note that the unselected depots are treated as non-depot vertices in V .

We obtain a new augmented graph $G' = (V \cup D', E \cup E')$ from G as follows. For each depot node u in D , we add a copy u' of u to G' , and create two new edges from u to u' , each with cost equal to half of the opening cost of u . We define the downwards monotone function f for the location-routing problem on G' as: $f(S) = 1$ if and only if $S \subseteq V$ and 0 otherwise. With this function, every cycle in the optimal solution of (IP) on G' must have an edge connecting a depot vertex u to its copy u' in D' . This corresponds to opening the depot u in G .

k -cycle covering problem: In this problem, we are given an integer k and an undirected complete graph $G = (V, E)$, where the edge costs satisfy the triangle inequality. The k -cycle covering problem is to find a cycle cover of G with minimum total cost, where each cycle in the cycle cover contains at least k vertices. For this problem, the corresponding downwards monotone function f can be defined as: $f(S) = 1$ if and only if S has less than k vertices, and 0 otherwise.

The location-routing problem with one depot is very similar to the classical traveling salesman problem, and therefore is NP-hard. The k -cycle covering problem is polynomial time solvable for $k \leq 3$, and is NP-hard for $k \geq 4$ (Imielinska et al. in [6] for $k = 4$, Vornberger in [9] for $k = 5$, and Cornuejols and Pulleyblank in [2] for $k \geq 6$). In the literature there are three 2-approximation algorithms for network design problems with downwards monotone functions: the GW-algorithm [3,10], the lightest-edge-first algorithm [4,6], and the heaviest-edge-first algorithm [5,8].

1.1 Our Contributions

The GW-algorithm is a generalized technique for approximating network design problems. It is a powerful tool; it can solve any problem that can be formulated as (IP) with downwards monotone functions, proper functions, or uncrossable functions, etc [5]. However, in the following we show that in some cases this becomes one of its weaknesses.

In this paper, we consider adding an extra constraint, which we call *the cardinality constraint*, to network design problems with downwards monotone

functions. This constraint is on the number of connected components in the optimal solution. More specifically, given an integer p , we require that there should be at most p connected components in the optimal solution. For example, when the cardinality constraint is imposed for the k -cycle covering problem, it means that not only each cycle in the cycle cover should contain at least k vertices, but also there should be at most p cycles in the cycle cover. It is easy to see that the new constraint has applications in vehicle routing, where only p vehicles are available to service the customers. For ease of explanation, we abbreviate the problems with the new cardinality constraint as *the p -constrained path/tree/cycle covering problems*. Here each connected component in the final solution is required to be a path/tree/cycle respectively.

In order to use the GW-algorithm, we need to define the function f to formulate the problem as (IP). When defining the function f for the p -constrained path/tree/cycle cover problems, given a set S of vertices we need to know the edges incident on the vertices in S , since we have to count the number of connected components involving these vertices. However, the domain of f in (IP) is the power set of V . Moreover, the function f needs to be pre-specified in order to formulate the problem as (IP), and we cannot define such a function when the input is still unknown. Therefore the GW-algorithm cannot be applied for the p -constrained path/tree/cycle covering problems as they cannot even be formulated as (IP).

In this paper we generalize the heaviest-edge-first algorithm and show that it is a 2-approximation for the p -constrained tree/cycle covering problems, and a 4-approximation for the p -constrained path covering problems. In order to achieve this, we first present a new combinatorial analysis of the approximation ratio for the heaviest-edge-first algorithm. This analysis is different from that in [5] which uses primal-dual approximation framework. We are then able to tackle the p -constrained path/tree/cycle covering problems, and show the performance bound of 2 for the p -constrained tree/cycle covering problems and the performance bound of 4 for the p -constrained path covering problems. We assume for the p -constrained path/cycle covering problems that the edge costs of the graph G satisfy the triangle inequality property.

2 Our Proposed Algorithm and Analysis

The input to our algorithm is an undirected graph $G = (V, E)$ with a non-negative edge cost $c(e)$ defined for every edge $e \in E$, and a constraint set C . The set C includes all the constraints of a network design problem modeled by downwards monotone functions, and the cardinality constraint. The output of our algorithm is a forest F' which satisfies all the constraints in C , where the cost of F' is within twice the cost of the optimal solution. The algorithm given in Fig. 1 has two stages, the growing stage and the deleting stage, which is the same as in the GW algorithm.

Since the feasibility test is straightforward, the running time of this algorithm is dominated by the minimum spanning tree construction. The fastest minimum

Input: An undirected graph $G = (V, E)$, non-negative edge costs, a constraint set C

Output: A forest F' on G , with a cost no more than twice the optimum

```

1 Comment: the growing stage, starts from an MST
2  $F \leftarrow$  any MST of  $G$ 
3 Comment: the deleting stage
4  $F' \leftarrow F$ 
5 for  $i=1$  to  $|V| - 1$ 
6    $e \leftarrow$  the edge with the  $i$ th largest cost in  $F$ 
7   if  $F' - \{e\}$  feasible then
8      $F' \leftarrow F' - \{e\}$ 
9 endfor

```

Fig. 1. Main algorithm for the p -constrained path/tree/cycle covering problems

spanning tree algorithm to date was developed by Chazelle [1]. Its running time is $O(|E|\alpha(|E|, |V|))$ where $\alpha(|E|, |V|)$ is the inverse Ackermann function.

In the following we define P to be a network design problem with downwards monotone functions, and define P' to be the problem obtained by adding the cardinality constraint to P . We say P and P' are the path/tree/cycle version if the connected components in the optimal solutions are required to be paths/trees/cycles respectively. We define $F'(P)$ and $F'(P')$ to be the solution produced by the algorithm in Fig. 1 for P and P' (tree version) respectively. Similarly we define $F^*(P)$ and $F^*(P')$ to be the optimal solutions of P and P' respectively.

2.1 Structure of the Solution $F'(P)$

We define the intersection $F'(P) \cap F^*(P)$ as follows. The intersection $F'(P) \cap F^*(P)$ includes only the edges that are present in both $F'(P)$ and $F^*(P)$. Note that each connected component of $F'(P) \cap F^*(P)$ is a minimum spanning tree of the vertices it spans.

If we represent each connected component of $F'(P) \cap F^*(P)$ as a super node, we obtain a new graph $G' = (V', E' \cup E'')$ as shown in Fig. 2(a). Each vertex in V'

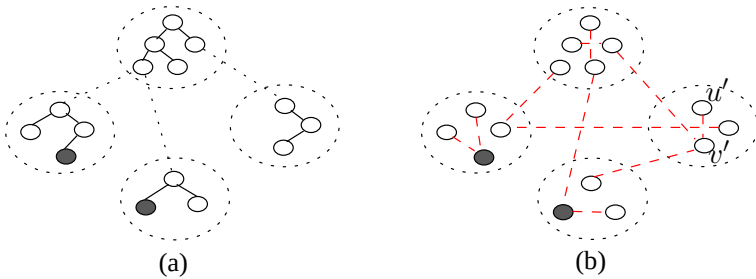


Fig. 2. The structure of $F'(P)$

is a super node corresponding to a connected component of $F'(P) \cap F^*(P)$. The solid edges are from $F'(P) - F^*(P)$, and they form the set E' . The dotted edges represent the edges of $F - F'(P)$, and form the set E'' . In Fig. 2(a) a connected component T' of $F'(P)$ is surrounded by a dotted circle. Define a set as active if and only if $f(S) = 1$, and as inactive otherwise. Clearly T' cannot have two or more disjoint inactive subtrees. Otherwise, an edge on the path connecting two disjoint inactive subtrees can be removed without affecting the feasibility of $F'(P)$. Define a subtree T_u of T' as minimal inactive, if T_u is inactive and every subtree of T_u is active. Since there are no two disjoint inactive subtrees in T' , the minimal inactive subtree of T' (rooted at, say, r'), if there is any, is uniquely defined. If T' is inactive, we can re-root T' at r' . Therefore T' has the property that each subtree of T' is active, which implies that T' has at most one inactive super node, and if one such node exists, it must be the root. We represent inactive super nodes by solid small circles in Fig. 2(a). In Fig. 2(b), the vertices are the super nodes in V' , and the dashed lines represent the edges in $F^*(P) - F'(P)$.

2.2 Performance Guarantee of the Solution $F'(P)$

According to the structure of $F'(P)$ in Fig. 2, all super nodes of a connected component T' in $F'(P)$, except possibly the root, are active. Each active super node must be connected to another super node by an edge of $F^*(P)$ (see Fig. 2(b)). We store in $H^*(P)$ two copies of each edge of $F^*(P)$ which connects a pair of super nodes of V' . These edges are the dashed edges in Fig. 2(b). Note that all the missing edges of $F^*(P)$, not present in $F'(P)$, are duplicated in $H^*(P)$. Our method of proving the approximation bound is to find a distinct edge e^* in $H^*(P)$ for each active super node u' , except the root super node, such that the cost of e^* is no smaller than the cost of the edge connecting u' to its parent in G' . If this is true, then we can claim that the approximation ratio of the algorithm in Fig. 1 is 2.

We fix some notations which will be used throughout this paper. We define T' to be a connected component of $F'(P)$ which will be the component of study in our analysis. For a super node u' , we denote the subtree rooted at u' in G' by $T_{u'}$. We use $(u', p(u'))$ to denote the edge from a super node u' to its parent node $p(u')$ in G' . We call a super node u' compensated by an unused edge e^* of $H^*(P)$ if $c(e^*) \geq c((u', p(u')))$. We define an edge of $H^*(P)$ to be of type-1 if its end points u' and v' are both in T' . We also define an edge of $H^*(P)$ to be of type-2 if its two end points are in different connected components of $F'(P)$. An example illustrating type-1 and type-2 edges is given in Fig. 3(a).

We describe two lemmas characterizing some relationships between the edges of G' and $H^*(P)$.

Lemma 1. *Let a type-2 edge e_1^* of $H^*(P)$ be incident on a super node u' . If for an ancestor super node $p(v')$ (parent of v') of u' in T' , $T' - T_{v'}$ is inactive, then $c(e_1^*)$ is no smaller than that of any edge on the path from u' to $p(v')$ in $F'(P)$.*

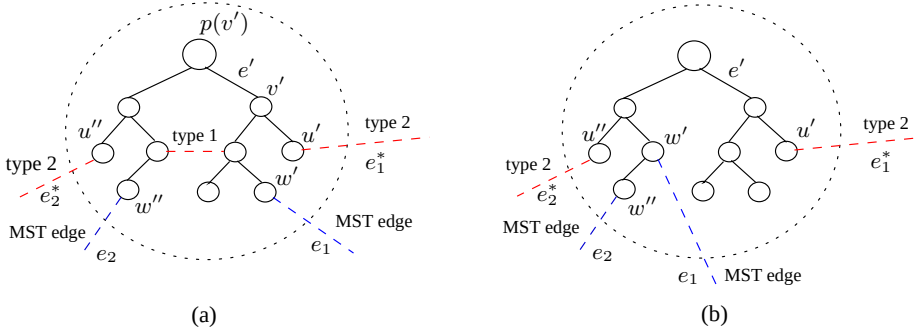


Fig. 3. Two types of edges

Proof. Let $e' = (v', p(v'))$ be an edge on the path from u' to the root r' in T' (Fig. 3(a)). Assume that e_1^* is not an element of F , therefore adding e_1^* to F creates a cycle. Let e_1 be an edge of E'' on the cycle that got deleted in the reverse deleting stage, and let e_1 be incident on w' in T' . If e' is on the path between u' and w' , then we are done since e' must be on the cycle, and according to the property of the minimum spanning tree, e_1^* has a cost no smaller than that of any edge on the cycle. Otherwise, consider the step just before e_1 to be deleted in the deleting phase. During that time, e' and e_1 are both in $F'(P)$. According to the property of downwards monotone functions, e' is also a candidate for deletion. However, the algorithm chooses to delete e_1 , thus we have $c(e_1^*) \geq c(e_1) \geq c(e')$. Setting $e_1^* = e_1$ and $u' = w'$, the case when $e_1^* \in F - F'(P)$ can be similarly argued.

Lemma 2. *Let the set $S \subseteq E''$ contain the edges incident on some nodes of T' which were deleted from F in the deleting phase of our algorithm. For any two edges e_1 and e_2 in S , $\max(c(e_1), c(e_2))$ is no smaller than the cost of any edge on the path which connects e_1 to e_2 in T' . Similarly, for two type-2 edges, e_1^* and e_2^* of $H^*(P)$, $\max(c(e_1^*), c(e_2^*))$ is no smaller than the cost of any edge on the path which connects e_1^* to e_2^* in T' .*

Proof. Let e' be an edge in T' on the path connecting e_1 to e_2 (Fig. 3(a)). Without loss of generality assume that $c(e_1) \leq c(e_2)$. Consider the step when e_2 is being considered for deletion from F . It is obvious that both e' and e_1 are available in F , thus deleting e' will also give us a feasible solution. In other words, e' is also a candidate edge for deletion. Since the algorithm deletes e_2 instead of e' , according to our deleting phase, we must have $c(e_2) \geq c(e')$.

We now prove the second claim. Assume that e_1^* and e_2^* are not in F . Adding e_1^* and e_2^* will create two cycles to $F'(P)$. Consider the following cases:

Case 1: The two cycles pass through some common vertex w' in T' . In this case it is easy to verify that e' must belong to one of the two cycles (Fig. 3(b)).

Case 2: The two cycles created by adding e_1^* and e_2^* are disjoint (Fig. 3(a)). Let them pass through two super nodes w' and w'' in T' respectively, and let w' and

w'' be incident to two edges e_1 and e_2 in E'' respectively. In this case e' must either be on the path from w' to w'' , or on one of the cycles. The claim holds as we have shown above that $c(e') \leq \max(c(e_1), c(e_2))$ if e' lies on the path from w' to w'' .

Therefore the lemma holds if e_1^* and e_2^* are not in F . The other cases can be similarly argued.

If we consider the edges of $H^*(P)$ incident on the super nodes of T' only, then we may get several connected subgraphs of $F^*(P)$ (see Fig. 2(b)). Let T_j^* be one of them. Note that T_j^* is a connected subgraph of $F^*(P)$. Define a super node u' as extreme with respect to T_j^* (or a set S of super nodes) if no ancestors of u' in T' exist in T_j^* (or S); u' is non-extreme otherwise. An example is given in Fig. 4(b). In Fig. 4(b) both u' and v' are extreme super nodes, but w' is non-extreme. The following lemma shows that all super nodes, except one extreme super node of T_j^* , can be compensated by edges of T_j^* (i.e., edges of $H^*(P)$). Recall that a super node u' is compensated if and only if an edge e^* of $H^*(P)$ is uniquely associated with u' , and $c(e^*) \geq c((u', p(u')))$.

Lemma 3. *Let T_j^* be a connected subgraph of $F^*(P)$ where only the super nodes of T' are involved. Only one (arbitrary) extreme super node of T_j^* cannot be compensated by edges of T_j^* .*

Proof. We prove this lemma by induction on the number of super nodes. The lemma is trivially true when T_j^* contains only one super node. Assume the lemma holds for all T_j^* with less than m super nodes. Given a T_j^* with m super nodes, we have two cases (Fig. 4):

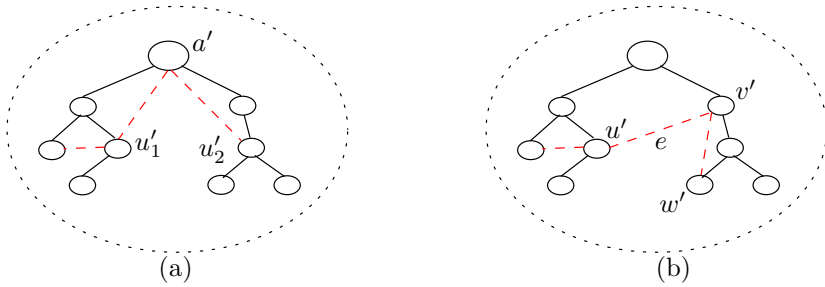


Fig. 4. Only one extreme super node cannot be compensated by edges in $F^*(P)$. The dashed edges (elements of $H^*(P)$) are the only edges of T_j^* .

Case 1: There exists only one extreme super node a' in T_j^* (Fig. 4(a)). We delete a' and all the edges of T_j^* incident on a' . Let $T_{j_1}^*, T_{j_2}^*, \dots, T_{j_t}^*$ be the resulting connected components of T_j^* and $u'_1, u'_2, \dots, u'_t$ be the extreme super nodes of such connected components (with respect to each of them). We need to show that only a' is not compensated and all $u'_l, l = 1, 2, \dots, t$ are compensated. According to our assumption, all super nodes except $u'_1, u'_2, \dots, u'_t$ can be compensated

by the edges in $T_{j_1}^*, T_{j_2}^*, \dots, T_{j_t}^*$. Since T_j^* is connected, each new connected component must have a different edge to a' . For a connected component $T_{j_l}^*$ ($1 \leq l \leq t$), let $e'_l = (a', v'_l)$ be such an edge. As u'_l is an arbitrary extreme super node of $T_{j_l}^*$, we can assume that u'_l is v'_l or an ancestor of v'_l . Adding e'_l to $F'(P)$ will create a cycle that includes the edge $(u'_l, p(u'_l))$, therefore u'_l can be compensated by e'_l .

Case 2: There exist at least two extreme super nodes, say u' and v' , in T_j^* . The two nodes u' and v' are connected in T_j^* (Figure 4(b)). Without any loss of generality we can assume that on the path $u' \sim v'$ there exists an edge e^* connecting u' or a descendant of u' to v' or to a descendant of v' . Therefore adding e to F creates a cycle that includes both $(u', p(u'))$ and $(v', p(v'))$. Deleting e^* from T_j^* , we get two connected components $T_{u'}^*$ and $T_{v'}^*$, each with fewer than m super nodes. According to our assumption, all super nodes of T_j^* except u' and v' can be compensated by edges of $T_{u'}^*$ and $T_{v'}^*$. Furthermore, $c(e^*) \geq c(u', p(u'))$ and $c(e^*) \geq c(v', p(v'))$. So in this case, all super nodes of T_j^* except u' or v' can be compensated by edges of T_j^* .

Let $S_C = \{T_1^*, T_2^*, \dots, T_l^*\}$ be any subset of the set of connected subgraphs of $F^*(P)$ where only the super nodes of T' are involved. We say a super node a in a component of S_C is an extreme super node of S_C , or a is extreme with respect to S_C , if there is no super node in the components of S_C that lies on the path from a to r' in T' . Suppose that for each T_i^* there exists a type-2 edge e_i^* attached to it. Let $E_C^* = \{e_1^*, e_2^*, \dots, e_l^*\}$. The following lemma can be viewed as a generalization of Lemma 3.

Lemma 4. *Only one (arbitrary) extreme super node of S_C cannot be compensated by the edges in E_C^* and the edges in the components of S_C . Moreover, at least one edge of E_C^* is not used to compensate any super node involved in S_C .*

Proof. We prove this lemma by induction on the number of connected components in S_C . Due to Lemma 3, it is trivially true if S_C only contains one connected component. Assume it's true for all such sets with fewer than l connected components. Consider a set S_C consisting of exactly l connected components. Without any loss of generality, assume T_1^* in S_C has an arbitrary super node a'_1 which is extreme with respect to S_C . Let $S'_C = \{T_2^*, \dots, T_l^*\}$. According to our assumption, there is only one arbitrary extreme super node a'_2 of S'_C that cannot be compensated, and one edge, say e_2^* of E_C^* unused. Let e_1^* and e_2^* be incident on u'_1 and u'_2 , where u'_1 is in T_1^* and u'_2 is in T_2^* . Without loss of generality, assume that $c(e_1^*) \geq c(e_2^*)$. Consider the following cases:

Case 1: T_1^* is interleaving with a component T_j^* in S'_C , in the sense that the path segment of T' connecting two end points of a type-1 edge e^* of T_1^* contains an extreme super node a'_j in T_j^* (Fig. 5(a)). Without loss of generality we assume that a'_j is extreme with respect to S'_C . Since a'_2 is assumed to be an arbitrary extreme super node, we can assume that $T_2^* = T_j^*$ and $a'_2 = a'_j$. Therefore a'_2

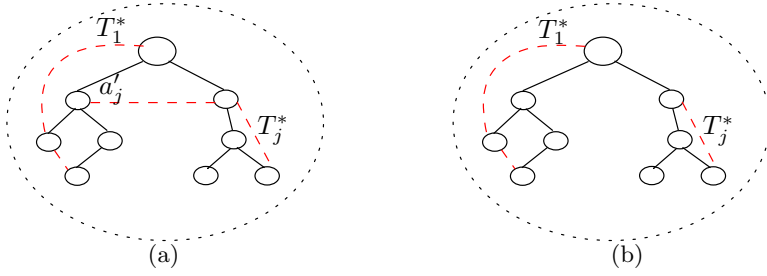


Fig. 5. T_1^* is interleaving with T_j^* in (a), but not in (b)

can be compensated by e^* , and for the components in S_C , only a'_1 has not been compensated, and we have two type-2 edges e_1^* and e_2^* unused.

Case 2: T_1^* is not interleaving with any components in S'_C . As a'_1 is extreme with respect to S_C , a'_2 is either a descendant of a'_1 or in a different branch from a'_1 . In both cases the edge $(a'_2, p(a'_2))$ must be on the path from e_1^* to e_2^* . By Lemma 2, the cost of $(a'_2, p(a'_2))$ is less than that of e_1^* . Therefore we can use e_2^* to compensate $(a'_2, p(a'_2))$, and for the components in S_C , only a'_1 has not been compensated, and we have a type-2 edge e_2^* unused.

We are now ready to prove the performance guarantee of the algorithm in Fig. 1 for network design problems with downwards monotone functions. The proof below is for the tree version. The cycle and path versions will be discussed later.

Theorem 1. *The cost of the forest $F'(P)$ found by the algorithm in Fig. 1 is bounded by twice the cost of the optimal solution $F^*(P)$ of P (tree version).*

Proof. Let $S_C = \{T_1^*, T_2^*, \dots, T_l^*\}$ be the set of connected subgraphs of $F^*(P)$ where only the super nodes of T' are involved. Suppose there exists a set $E_C^* = \{e_1^*, e_2^*, \dots, e_l^*\}$ where a type-2 edge e_i^* is attached to T_i^* . The theorem then holds, since due to Lemma 4 only one (arbitrary) extreme super node a' of S_C cannot be compensated by the edges of these components, and an edge, say e_1^* , of E_C^* is unused. In the following we assume that there exists a connected component, say T_0^* , of $F^*(P)$ whose super nodes are all from T' but no type-2 edges of $F^*(P)$ are attached to T_0^* .

As shown in Lemma 3, only one extreme super node a'_0 in T_0^* cannot be compensated by the edges of T_0^* . If T_0^* does not contain the root r' of T' , then T_0^* must have at least two extreme super nodes since each subtree of T' is active. We can pick an additional copy of an edge e^* of T_0^* to compensate a'_0 (please refer to our proof of Case 2 for Lemma 3). In this case as r' is in one of the components in S_C , we must have $a' = r'$. The theorem then holds as only r' is not-compensated and an edge e_1^* of E_C^* remains unused. The case when T_0^* includes r' can be argued similarly as in the proof for Case 1 and Case 2 in Lemma 4.

Thus for each super node u' in T' , an edge in $F^*(P)$ can be found to have equal or larger cost than that of an edge in $F'(P)$ connecting u' to its parent in T' . In all cases, each edge in $F^*(P)$ is used at most twice to compensate its two end points. Clearly we have that the cost of $F'(P)$ is at most twice that of $F^*(P)$.

The following theorem states that the approximation ratio is still 2 when P is of cycle version.

Theorem 2. *The cost of the forest $F'(P)$ found by the algorithm in Fig. 1 is at most the cost of the optimal solution $F_c^*(P)$ of P (cycle version).*

Proof. For a cycle in the optimal solution, the number of edges and the number of super nodes in the cycle are the same. For each super node u' , we try to find a distinct edge e^* in the optimal solution $F_c^*(P)$ to compensate u' . Consider a connected subgraph C^* of $F_c^*(P)$ whose super nodes are all from a connected component T' of $F'(P)$, we have the following cases:

Case 1: C^* is a cycle. When P is of cycle version, a super node u' may have a self-loop edge e^* in F_c^* . For the k -cycle covering problem, this happens when u' corresponds to a path with at least k vertices of G . In this case u' together with e^* represents a cycle in F_c^* . As u' is inactive, it must be the root r' , but r' needs not to be compensated. Therefore in the following, we assume that C^* does not contain self-loop edges.

If C^* does not contain the root, then C^* must have two extreme super nodes u' and v' which are in different branches of r' , as T' is rooted in such a way that each branch of T' is active. u' is connected to v' in C^* , so without any loss of generality we can assume that there exists an edge e^* in C^* with one end in $T'_{u'}$ and the other end in $T'_{v'}$. If we remove e^* from C^* , then C^* becomes a path. According to Lemma 3, only one extreme super node, say u' , cannot be compensated by the edges on the path. The theorem then holds, as e^* can be used to compensate u' .

Case 2: C^* is a path. In this case, we consider all the paths involving the super nodes of T' . Let $S_C = \{P_1^*, P_2^*, \dots, P_l^*\}$ be the set of all such paths, and let $E_C^* = \{e_1^*, e_2^*, \dots, e_l^*\}$ be a set of type-2 edges of F_c^* , where each edge e_i^* of E_C^* , $1 \leq i \leq l$, is attached to P_i^* . Note that for each path P_i^* , where $1 \leq i \leq l$, e_i^* is picked arbitrarily from the two type-2 edges connecting P_i^* to outside T' . Applying Lemma 4 on S_C and E_C^* , only one involved extreme super node a' of S_C cannot be compensated, and one edge e^* of E_C^* remains unused.

If the root is in one of the paths, then we are done, since the root needs not to be compensated. Otherwise, the root is in a cycle C_r^* without type-2 edges. The remaining proof follows in the same way as that in the proof for Theorem 1.

Therefore we have shown that the cost of $F'(P)$ is at most the cost of $F_c^*(P)$. Since we double the edges in $F'(P)$ to get our final cycle solution, the approximation ratio of our algorithm for P (cycle version) is still 2.

Theorem 1 also holds when P is of path version. By doubling the edges in $F^*(P)$ we get a cycle cover C of the underlying graph G . A path cover can be obtained

after removing an arbitrary edge from each cycle in C . Therefore we have the following theorem:

Theorem 3. *The algorithm in Fig. 1 is a 4-approximation for P (path version).*

2.3 Performance Guarantee for the p -Constrained Path/Tree/Cycle Covering Problems

We claim that the algorithm in Fig. 1 is a 2-approximation for P' , a p -constrained tree/cycle covering problem. The algorithm in Fig. 1 for P' will stop deleting edges from the minimum spanning tree if the number of components in the remaining forest is already p .

Assume c connected components exist in $F'(P)$. It is easy to see that $F'(P')$ can be obtained from $F'(P)$, by greedily adding $c - p$ minimum spanning tree edges between the connected components of $F'(P)$. Let these $c - p$ edges form the set S . The edges in S represent the optimal way to reduce the number of connected components in $F'(P)$ to p . However, to prove the performance guarantee for P' , we need to locate $c - p$ edges in $F^*(P')$ which have a total cost no smaller than that of the edges in S . Also note that these $c - p$ edges in $F^*(P')$ can be used at most once for compensating the super nodes of G' .

Our analysis utilizes the fact that in the proof of the performance guarantee for P (tree/cycle version), if the root r' of a connected component T' of F' is active, then there must exist a type-2 edge e^* incident on a super node of T' that is not used to compensate any super node in T' . In other words, we cannot use such a type-2 edge e^* to bound the cost of the edges in S when the root is inactive. In order to prove the ratio for $F'(P')$, our analysis starts from the solution $F'(P)$, and focuses on locating a set S^* of $c - p$ edges in $F^*(P')$ between the connected components of $F'(P)$. In the next theorem, we locate such a set S^* and show that the edges in S^* are guaranteed to be used at most once to compensate some super nodes of G' . We give the formal proof in Theorem 4.

Theorem 4. *The cost of the forest $F'(P')$ found by the algorithm in Fig. 1 for P' (a p -constrained tree covering problem), is bounded by twice the optimum.*

Proof. Note that all the above analysis for problem P (tree/cycle version) can also be applied for $F^*(P')$, the optimal solution for P' . This is due to the fact that the constraints for P are also present in P' .

Let T' be a connected component of $F'(P)$. If the root r' of T' is active, then we are done as in the proof of Theorem 1, it is maintained that if r' is active, then there must exist an unused type-2 edge e^* (with one end in T') in $H^*(P')$. Here $H^*(P')$ contains two copies of the edges in $F^*(P')$. The edge e^* would have a cost of at least the smallest minimum spanning tree edge to connect T' to another connected component in $F'(P)$.

In the following we assume that r' is inactive. Consider the case when there exists an edge e^* of $H^*(P')$ incident on r' . If e^* is of type-2, then we are also done, as a copy of such an edge has not been used to compensate any super node in T' in the analysis for $F'(P)$ (tree/cycle version). Otherwise e^* connects r' to

another super node u' also in T' . But this corresponds to Case 1 in the proof of Theorem 1, where we can also get one unused type-2 edge e_1^* in $H^*(P')$ incident on a super node of T' .

Therefore whenever r' is incident to an edge e^* in $H^*(P')$, e^* or another type-2 edge e_1^* of $H^*(P')$ can be used to connect T' to another connected component in $F'(P)$. Let c_1 denote the number of connected components of $F'(P)$ whose roots are not incident to any edge in $H^*(P')$. The number of connected components of $F^*(P')$ is at least c_1 , as each super node not incident to any edge of $H^*(P')$ is a connected component by itself. Thus $p \geq c_1$. It is equivalent to say that the number of connected components of $F'(P)$ whose roots are incident to some edges in $H^*(P')$, is $c - c_1 \geq c - p$. Therefore we have located at least $c - p$ edges in $H^*(P')$ to allow the merge of the connected components in $F'(P)$. This completes the proof.

Similarly we can establish the following theorem.

Theorem 5. *The algorithm in Fig. 1 is a 2-approximation for the p -constrained cycle covering problem and a 4-approximation for the p -constrained path covering problem.*

3 Conclusions

In this paper, we present a different combinatorial analysis for the algorithm in [5] for network design problems with downwards monotone functions. The GW-algorithm is inadequate to handle the cardinality constraint where the number of connected components in the optimal solution should be no more than p for some integer p . We show that the algorithm in [5] can be generalized to further deal with this constraint. Our analysis shows that the approximation ratios for network design problems with downwards monotone functions remain the same after introducing the cardinality constraint. In the future, we would like to investigate the possibility of improving the approximation ratio for the k -cycle covering problem based on this analysis.

References

1. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM* 47, 1028–1047 (2000)
2. Cornuejols, G., Pulleyblank, W.: A matching problem with side constraints. *Discrete Mathematics* 29, 135–159 (1980)
3. Goemans, M.X., Williamson, D.P.: A general approximation technique for constrained forest problems. In: *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pp. 307–316 (1992)
4. Goemans, M.X., Williamson, D.P.: Approximating minimum-cost graph problems with spanning tree edges. *Operations Research Letters* 16, 183–194 (1994)
5. Goemans, M.X., Williamson, D.P.: The primal-dual method for approximation algorithms and its application to network design problems. In: Hochbaum, D. (ed.) *Approximation Algorithms for NP-hard Problems*, pp. 144–186 (1997)

6. Imielinska, C., Kalantari, B., Khachiyan, L.: A greedy heuristic for a minimum-weight forest problem. *Operations Research Letters* 14, 65–71 (1993)
7. Laporte, G.: Location-routing problems. In: Golden, B.L., Assad, A.A. (eds.) *Vehicle routing: Methods and studies*, Amsterdam, pp. 163–197 (1988)
8. Laszlo, M., Mukherjee, S.: An approximation algorithm for network design problems with downwards-monotone demand functions. *Optimization Letters* 2, 171–175 (2008)
9. Vornberger, O.: Complexity of path problems in graphs, PhD thesis, Universitat-GH-Paderborn (1979)
10. Williamson, D.P., Goemans, M.X., Mihail, M., Vazirani, V.V.: A primal-dual approximation algorithm for generalized Steiner network problems. In: *Proceedings of the 25th annual ACM-SIAM symposium on Discrete algorithms*, pp. 708–717 (1993)

An FPTAS for the Minimum Total Weighted Tardiness Problem with a Fixed Number of Distinct Due Dates^{*}

George Karakostas^{1, **}, Stavros G. Kolliopoulos², and Jing Wang³

¹ McMaster University, Dept. of Computing and Software, 1280 Main St. West,
Hamilton, Ontario L8S 4K1, Canada

karakos@mcmaster.ca

² Dept. of Informatics and Telecommunications, National and Kapodistrian
University of Athens, Athens 157 84, Greece

sgk@di.uoa.gr

³ McMaster University, School of Computational Engineering & Science, 1280 Main
St. West, Hamilton, Ontario L8S 4K1, Canada

wang257@mcmaster.ca

Abstract. Given a sequencing of jobs on a single machine, each one with a weight, processing time, and a due date, the tardiness of a job is the time needed for its completion beyond its due date. We present an FPTAS for the basic scheduling problem of minimizing the total weighted tardiness when the number of distinct due dates is fixed. Previously, an FPTAS was known only for the case where all jobs have a common due date.

1 Introduction

The minimum total weighted tardiness problem for a single machine is defined as follows. We are given n jobs, each with a weight $w_j > 0$, processing time p_j , and due date d_j . When these jobs are sequenced on a single machine, each job j will have a completion time C_j . The tardiness T_j of job j is defined as $\max\{0, C_j - d_j\}$. If $T_j = 0$, the job is *early*, otherwise it is *tardy*. The objective is to minimize the total weighted tardiness, i.e., minimize $\sum_j w_j T_j$.

The problem is very basic in scheduling (see surveys [1],[10] and the references in [4],[5]) and is known to be NP-hard [8] even in the case of unit weights [3]. Despite the attention it has received, frustratingly little is known on its approximability. The best known approximation algorithm has a performance guarantee of $n-1$ [2]. For the unit weight case, Lawler gave early on a fully polynomial-time approximation scheme (FPTAS) [7], which is a modification of his pseudopolynomial dynamic programming algorithm in [6]. For general weight values, the problem remains NP-hard even when all jobs have a common due date [11].

^{*} Full version at http://www.cas.mcmaster.ca/~gk/papers/tardiness_fixed.pdf

^{**} Research supported by an NSERC Discovery grant.

Kolliopoulos and Steiner [5] gave a pseudopolynomial dynamic programming algorithm for the case of a fixed number of distinct due dates. Using essentially Lawler's rounding scheme from [7], they obtained an FPTAS only for the case of polynomially bounded weights. Kellerer and Strusevich [4] gave an FPTAS for general weights in the case where all jobs have a common due date. The existence however of an FPTAS for the case of general weights and a fixed number of distinct due dates has remained open. For a general number of distinct due dates the problem becomes strongly NP-hard [6].

In this work, we settle the case of a fixed number of distinct due dates by giving an FPTAS. We design first a pseudopolynomial algorithm and then apply the rounding scheme of [4] to obtain the desired approximation scheme. We exploit two crucial properties of the algorithms in [4]. The first is that the optimal choice is feasible at every job placement the FPTAS performs (cf. Lemma 9). This step-by-step mimicking of the optimal chain of computation is crucial for bounding the approximation error. Of course, the schedule we output may be suboptimal due to our approximate ("rounded") estimation of tardiness. The second property is that the rounding scheme of [4] produces values which correspond to actual schedules; therefore by rounding up the processing time of tardy jobs with due date d , one rounds down the processing time of early jobs with the same due date *by the same amount*. Since the total time needed for these jobs remains the same, this means that there is empty space that allows our algorithm to push back the extra tardy processing time towards the past. This need for *preemption*, i.e., allowing the processing of a job to be interrupted and later restarted, did not arise in [4] where the extra tardy processing time past the common due date D could be accommodated in the time interval $[D, \infty)$.

In addition to these basic facts, we need a number of other new ideas. Our algorithm works in two stages. First, via dynamic programming it computes an assignment of the job completion times to the time horizon, where only a subset of the jobs is explicitly packed and the rest are left "floating" from their completion time backwards. This is what we call an *abstract schedule*. In the second stage, a greedy procedure allocates the actual job lengths, possibly also with preemption. As in previous algorithms, the jobs that straddle a due date in a schedule, the so-called *straddlers*, play an important role. We observe that only the placement of the tardy straddlers is critical. The time intervals, called *superintervals*, between consecutive tardy straddlers, form the basic time unit on our time horizon. The scheduling of a job j as early as then can be localized within only one of these superintervals, depending on the actual d_j value (cf. Lemma 3). This helps to shrink the state space of the dynamic program.

It is well-known that the preemptive and non-preemptive optima coincide when minimizing tardiness on a single machine [9]. This powerful fact has found only limited use in approximation algorithms so far, for example through the preemptive scheduling of early jobs in [5]. We take the opposite view from [5] and insist on the non-preemptive scheduling of early jobs. Moreover, all early jobs are packed explicitly in the abstract schedule. This is necessary since early jobs are particularly difficult to handle: enumerating their total length is prohibitive

computationally and distorting their placement even by a tiny amount might result in a severely suboptimal schedule. We allow instead preemptive scheduling of the tardy jobs. As explained above, preemption will allow us to flexibly push back the extra tardy processing time, introduced by the rounding, towards the past. Following this idea to its natural conclusion, we allow even straddlers to be preempted. In the final schedule, it could be that only the completion time of a tardy job happens in the interval in which it was originally assigned by the dynamic program, while all the processing happens earlier. The algebraic device we introduce that allows the abstract schedule to keep some of the jobs “floating”, without pinning down anything but their completion time, is the *potential* empty space within a prefix of a schedule (cf. Eq. (3) below). To ensure that preemptions can be implemented into *actual* empty space is perhaps the largest technical difficulty in our proof.

The approximability of total weighted tardiness problem with an arbitrary number of distinct due dates remains as the main open problem.

2 Structural Properties of an Optimal Schedule

We are given n jobs $j = 1, \dots, n$, each with its own processing time p_j and weight w_j and a due date from a set of K possible distinct due dates $\{d_1, d_2, \dots, d_K\}$, where K will be assumed to be a constant for the rest of this paper. For convenience, we are also going to define the artificial due date $d_0 = 0$. The due dates partition the time horizon into $K+1$ intervals $I_l = [d_{l-1}, d_l]$ for $l = 1, \dots, K$, and $I_{K+1} = [d_K, \infty)$. We partition the jobs into K classes $C_1, C_2, \dots, C_K$ according to their due dates.

A crucial concept for the algorithms we describe is the grouping of intervals I_l in the following manner: for any i_u, i_{u+1} , intervals $I_{i_u+1}, I_{i_u+2}, \dots, I_{i_{u+1}}$ are grouped into a *superinterval* $G_{i_u, i_{u+1}} = I_{i_u+1} \cup I_{i_u+2} \cup \dots \cup I_{i_{u+1}} = [d_{i_u}, d_{i_{u+1}})$, if straddlers S_{i_u} and $S_{i_{u+1}}$ are consecutive tardy straddlers, i.e., there is no other tardy straddler in between due dates $d_{i_u}, d_{i_{u+1}}$. Note that it may be the case that $i_{u+1} = i_u + 1$, i.e., $G_{i_u, i_{u+1}} \equiv I_{i_u+1}$ if both $S_{i_u}, S_{i_{u+1}}$ are tardy. Also, since straddler S_K is tardy, the last superinterval is $G_{K, K+1} = I_{K+1}$.

In any schedule of the n jobs, a job that finishes before or on its due date will be an *early* job, otherwise it will be *tardy*. We also call any job that starts before or on a due date but finishes after it a *straddler*. It is well-known [9] that the optimal values of the preemptive and the non-preemptive version of the problem are the same. Therefore we can assume that the optimal schedule is a non-preemptive one. In it the straddlers will appear as contiguous blocks, crossing one or more due dates. For easiness of exposition, we will assume that there is an optimal schedule with distinct straddlers for every due date, i.e., there are K distinct straddlers $S_1, \dots, S_K$ corresponding to due dates $d_1, \dots, d_K$. After the description of the algorithms, it should be clear how to modify them in order to deal with the special case of some straddlers crossing more than one due dates. For convenience, let also S_0 be an artificial tardy straddler for d_0 with $w_{S_0} = p_{S_0} = 0$. In any optimal schedule, the machine has clearly no idle

time. Hence, wlog, due dates that are greater than $\sum_j p_j$, can be set to ∞ . Accordingly, we can assume that there is a straddler for every due date.

Tardy straddlers are of particular interest. We will assume that we have guessed the number $M \leq K$ of tardy straddlers and these tardy straddlers $S_{i_1}, \dots, S_{i_M}$ of the optimal schedule (also $S_{i_0} = S_0$). By guessing, we mean the exhaustive enumeration of all combinations of jobs with due dates (with repetition in the general case where a job can be straddler of more than one due dates), which produces a polynomial number of possibilities, since K is constant. Let $m = n - M$ be the number of the remaining jobs, which are ordered according to their weighted shortest processing times (WSPT), i.e., $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \dots \leq \frac{p_m}{w_m}$. With some abuse of terminology, we will call these jobs *non-straddling*, although some of them are the early straddlers. We assume that we have guessed a bound Z^{ub} such that for the optimal value OPT we have $Z^{ub}/2 \leq OPT \leq Z^{ub}$. This can be achieved by enumeration of $O(\log(n^2 w_{max} p_{max}))$ values.

It should be obvious that, in any interval I_i , the tardy jobs in that interval are processed before the early ones. It is also well-known (e.g., see Lemma 2.1 in [5]) that the tardy jobs must be processed in WSPT order. With respect to a given partial schedule we define the following quantities, which are going to be important throughout this work:

- $y_k^{(i-1)t}$, $1 \leq t < i \leq K+1$, $1 \leq k \leq m$: the total processing time of those (tardy) jobs among the first k (in WSPT order) jobs, that belong to class C_t and are in I_i . Also define $y_k^{0t} = 0$ for all t .
- $W_k^{(i-1)t}$, $1 \leq t < i \leq K+1$, $1 \leq k \leq m$: the total weight of the jobs in the previous item.
- A_k^t , $1 \leq t \leq K$, $1 \leq k \leq m$: the total processing time of the class C_t jobs among the first k jobs. Notice that these quantities can be calculated in advance.
- e_k^{it} , $1 \leq i \leq t \leq K$, $1 \leq k \leq m$: the total processing time of those (early) jobs among the first k (in WSPT order) jobs, that belong to class C_t and are in I_i .

The following lemmas are important properties of an optimal schedule:

Lemma 1. *In the optimal schedule and for any $1 \leq i \leq K$, if S_i is tardy, then for any $1 \leq l \leq i$ and any $i+1 \leq u \leq K$, we have $e_k^{lu} = 0$.*

Lemma 2. *In the optimal schedule and for any $2 \leq i \leq K$, if S_{i-1} is early, then $y_k^{(i-1)u} = 0$ for any $1 \leq u \leq i-1$, i.e., there are no tardy jobs in I_i .*

Lemma 2 implies that the only non-zero y 's are the ones that correspond to the first interval of each superinterval. Therefore, from now on we will use only the values $y_k^{i_u t}$, $1 \leq u \leq M$, $1 \leq t \leq i_u$, $1 \leq k \leq m$. Lemmas 1 and 2 imply that for every $1 \leq k \leq m$ and for every $1 \leq t \leq K$ s.t. $i_{s-1} < t \leq i_s$ for some $1 \leq s \leq M$ we have

$$A_k^t = \sum_{u=s}^M y_k^{i_u t} + \sum_{q=i_{s-1}+1}^t e_k^{qt} \quad (1)$$

A direct consequence of Lemma 1 and the definition of a superinterval is the following.

Lemma 3. (Bracketing Lemma for early jobs) *Let $u \leq M$. In an optimal schedule only jobs from classes C_t , with $i_{u-1} < t \leq i_u$ can be assigned as early in the superinterval $G_{i_{u-1}i_u}$.*

3 Dynamic Programming to Find an Abstract Schedule

An *abstract schedule* is an assignment of the the m non-straddling jobs to superintervals so that (i) early jobs are feasibly and non-preemptively packed within their assigned superinterval (ii) there is enough empty space so that tardy jobs that complete in their assigned superinterval can be preemptively packed and (iii) there is enough empty space so that the M tardy straddlers can be preemptively packed. An *abstract k -schedule*, $k \leq m$, is an abstract schedule for the first k non-straddling jobs. In this section we describe a pseudopolynomial dynamic programming algorithm (DP) that computes a suitable abstract schedule. In the next section we show how to pin down the actual processing of the tardy jobs and the straddlers, so that the abstract schedule is converted to an actual schedule of the n jobs with minimum total tardiness.

The DP algorithm “guesses” the M tardy straddlers. Extending the dynamic programming of [4], the states of DP store the following values for a (partial) schedule of the k first (in WSPT order) of the m non-straddling jobs:

$$\left(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \dots, y_k^{i_M 1}, W_k^{i_M 1}, y_k^{i_1 2}, W_k^{i_1 2}, \dots, y_k^{i_M K}, W_k^{i_M K}\right), \quad (2)$$

where Z_k is the total weighted tardiness of the k scheduled jobs. Note that some of the $y_k^{i_u j}, W_k^{i_u j}$ in (2) may not exist, if $i_u < j$. As in [4], the weight values $W_k^{i_u j}$ will be needed when the tardy straddlers will be re-inserted at the end.

The initial state will be $(0, 0, \dots, 0)$. A *state-to-state transition* from state (2) corresponds to the insertion of the $(k + 1)$ -th job in a super-interval of the (partial) abstract schedule of the previous k jobs. Such a transition corresponds to the choice of inserting this job in a superinterval, and must be *feasible*. The feasibility conditions, described in detail below, require that there is enough empty space to insert the new job in the selected superinterval, and there is still enough empty space for the re-insertion of the straddlers. Note that the combination of the class C_t of the inserted job and the superinterval $G_{i_{u-1}i_u}$ chosen for it by the transition determines whether this job is early or tardy: if $1 \leq t \leq i_{u-1}$ then the job is tardy, otherwise it is early.

In order to check the feasibility of the transitions, we would like to be able to calculate the empty space in every superinterval from the information stored in states (2). Unfortunately, this is not possible, because essentially there are many possibilities for the placement of early jobs that yield the same state and keeping track of all these possibilities would blow up the state space. As a result of this limited information, some of the space that looks empty will be actually needed to accommodate preempted parts of tardy jobs from later superintervals. Nevertheless, we can calculate the potential empty space for *prefixes* of

the schedule that start from time $t = 0$. The processing time for a tardy job is just slated for the prefix that ends at its assigned completion time by the first (dynamic programming) stage of the algorithm, without pinning down its exact placement. This placement is fixed only during the second stage of the algorithm. We introduce the following set of prefix values:

- L_k^{0l} , $1 \leq l \leq K$, $1 \leq k \leq m$: the total space from d_0 to d_l minus the space taken by the jobs whose class indices are less than or equal to l .

Given $1 \leq l \leq K$, let s be such that $i_{s-1} < l \leq i_s$. Then L_k^{0l} can be computed from the information of state (2) as follows:

$$\begin{aligned} L_k^{0l} &= d_l - \left(\sum_{j=1}^{s-1} \sum_{q=i_{j-1}+1}^{i_j} \sum_{h=q}^{i_j} e^{qh} + \sum_{q=i_{s-1}+1}^l \sum_{h=q}^l e^{qh} \right) - \left(\sum_{j=1}^{s-1} \sum_{h=1}^{i_j} y^{i_j h} \right) \\ &= d_l - \sum_{i=1}^l A_k^i + \sum_{j=s}^M \sum_{h=1}^l y^{i_j h} \end{aligned} \quad (3)$$

Recall that there are M tardy straddlers $\{S_{i_u}\}_{u=1}^M$ overall. We assume that the $(k+1)$ -th job J_{k+1} belongs to class C_t , and that we want to schedule it in superinterval $G_{i_{u-1}i_u}$. Note that Lemma 3 implies that, to even consider such a placement, $t \leq i_u$ must hold. The three feasibility conditions that must be satisfied by a DP transition from state (2) follow. From equation (3), given the state information, all three can be effectively checked.

Condition (1): $t \leq i_{u-1}$, i.e., J_{k+1} is tardy.

- 1a. Check that $L_k^{0l} - L_k^{0i_{u-1}} \geq p_{k+1}$ holds $\forall l$ s.t. $i_{u-1} \leq l \leq i_u$.
- 1b. If 1a doesn't hold, check that $L_k^{0l} \geq p_{k+1}$ holds $\forall l$ s.t. $i_{u-1} < l \leq i_u$.
- 1c. Check that $L_k^{0i_j} \geq p_{k+1}$ holds $\forall j$ s.t. $u < j \leq M$.

Condition (2): $i_{u-1} < t \leq i_u$, i.e., J_{k+1} is early.

- 2a. Check that $L_k^{0l} - L_k^{0i_{u-1}} \geq p_{k+1}$ holds $\forall l$ s.t. $t \leq l \leq i_u$.
- 2b. If 2a doesn't hold, check the following according to which case applies:
 - 2b.1. $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \leq L_k^{0i_{u-1}}$: Check that $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^l \sum_{v=q}^l e_k^{qv}) \geq p_{k+1}$ and $L_k^{0l} \geq p_{k+1}$ hold $\forall l$ s.t. $t \leq l \leq i_u$;
 - 2b.2. $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} > L_k^{0i_{u-1}}$: Check that $L_k^{0l} \geq p_{k+1}$ holds $\forall l$, s.t. $t \leq l \leq i_u$.
- 2c. Check that $L_k^{0i_j} \geq p_{k+1}$ holds $\forall j$, s.t. $u < j \leq M$.

Condition (3): Check $L_{k+1}^{0j} \geq \sum_{h=1}^{u-1} p_{i_h}$, $\forall u, j$ s.t. $1 < u \leq M, i_{u-1} < j \leq i_u$.

Condition (3) will ensure that there is always enough empty space to fit the straddlers in the final schedule (Lemma 7). Conditions (1a) (and (2a)) are satisfied when there is enough space to fit J_{k+1} as tardy (or early) in a *non-preemptive* schedule. They are redundant if we are looking for a preemptive schedule. But we will use the fact that Conditions (1a),(2a),(3) are enough for the construction of an optimal DP algorithm which produces an optimal *non-preemptive* schedule in the analysis of our FPTAS (Sections 4, 5).

The new state $(k+1, Z_{k+1}, \dots)$ after the (feasible) insertion of the $(k+1)$ -th job J_{k+1} of class C_t in superinterval $G_{i_{u-1}i_u}$ is computed as follows:

- J_{k+1} is **early**: Set $Z_{k+1} = Z_k$, $y_{k+1}^{i_u j} = y_k^{i_u j}$, $W_{k+1}^{i_u j} = W_k^{i_u j}$ for all $1 \leq u \leq M$, $1 \leq j \leq i_u$.
- J_{k+1} is **tardy**: Set $Z_{k+1} = Z_k + w_{k+1}(\sum_{v=1}^{i_u-1} y_k^{i_u-1 v} + p_{k+1} + d_{i_u-1} - d_t)$, $y_{k+1}^{i_u-1 t} = y_k^{i_u-1 t} + p_{k+1}$, $W_{k+1}^{i_u-1 t} = W_k^{i_u-1 t} + w_{k+1}$. Note that we reject the insertion if $Z_{k+1} > Z^{ub}$, and if at some point we determine that this inequality is true for all possible insertions of J_{k+1} then we reject Z^{ub} , we replace it with a new $Z^{ub} := 2Z^{ub}$ and start the algorithm from scratch.

Lemma 4. *Let $u \leq M$, $1 \leq k \leq m$. If $L_k^{0i_j} \geq 0$, $\forall j$ s.t. $1 \leq j \leq u$, then there is enough actual empty space to pack preemptively the tardy jobs that have so far been assigned to the first u superintervals.*

Lemma 5. *Assume state (2) corresponds to an abstract k -schedule. Conditions (2) and (3) imply that job J_{k+1} is packed non-preemptively as early in the intervals $I_{i_{u-1}+1}, \dots, I_{i_u}$, so that we obtain an abstract $(k+1)$ -schedule. Moreover all early jobs complete as close to their due date as possible.*

Lemma 6. *Assume state (2) corresponds to an abstract k -schedule. Conditions (1) and (3) imply that one can assign job J_{k+1} to complete as tardy in the superinterval $G_{i_{u-1}i_u}$, so that we obtain an abstract $(k+1)$ -schedule.*

4 Producing an Optimal Schedule

Condition (1) only ensures that there is enough empty space to fit each tardy job, possibly broken in pieces. Now we describe the procedure that actually allocates the tardy jobs on the time horizon:

- The (tardy) jobs in the last interval $I_{K,K+1}$ are scheduled in that interval non-preemptively in WSPT order.
- For $u = M, M-1, \dots, 1$ look at the tardy jobs with completion times in $G_{i_{u-1}i_u}$, i.e., in interval $I_{i_{u-1}, i_{u-1}+1}$ in WSPT order. While there is empty space in this interval, fit in it as much processing time as possible of the job currently under consideration. If at some point there is no more empty space, the rest of the processing times of these tardy jobs will become preempted pieces to be fitted somewhere in $[d_0, d_{i_{u-1}})$. Then, we fill as much of the remaining empty space in $G_{i_{u-1}i_u}$ as possible using preempted pieces belonging to preempted tardy jobs in $[d_{i_u}, d_K]$ in WSPT order (although the particular order doesn't matter). When we run out of either empty space or preempted pieces, we move to the next $u := u-1$.

We note that the above process does not change the quantities L_m^{0j} , $j = 1, 2, \dots, K$, and therefore Condition (3) continues to hold.

The placement of the tardy straddlers will complete the schedule the algorithm will output. The following lemma shows how we will place the straddlers

preemptively so that two properties are maintained: (a) straddler S_{i_u} completes at or after d_{i_u} and before $d_{i_{u+1}}$, for all $u = 1, 2, \dots, M - 1$, and (b) the prefix of the schedule that contains all straddlers' processing time is *contiguous*, i.e., there are no 'holes' of empty space in it. We will need property (b) in the calculation of the total tardiness of the final schedule below and in our FPTAS. We emphasize that (b) may force us to preempt straddlers: for example, suppose that the empty space in $[d_0, d_1]$ is much bigger than $\sum_{h=1}^M p_{S_{i_h}}$; then our schedule will use $\sum_{h=1}^M p_{S_{i_h}}$ units at the beginning of that empty space to process $S_{j_1}, \dots, S_{j_M}$, while setting their completion times at $d_{j_1}, \dots, d_{j_M}$ respectively.

Lemma 7. *The placement of the tardy straddlers can be done so that properties (a),(b) above are maintained.*

Given that Z^{ub} is large enough, the dynamic programming will ultimately produce a set of states with their first coordinate equal to m , i.e., states that correspond to partial schedules of all m non-straddling jobs. Since these states satisfy Condition (3), Lemma 7 implies that we can re-insert the straddlers at their correct position without affecting the earliness of the early or the placement in intervals of the tardy non-straddling jobs, thus creating a number of candidate full schedules. Let $\{T_{i_u}\}_{u=1}^M$ be the tardiness of the M tardy straddlers. Also, note that due to property (b) in Lemma 7, $x_{i_u} := \max\{0, \sum_{l=1}^u p_{S_{i_l}} - L_m^{0i_u}\}$ is the part of S_{i_u} beyond due date d_{i_u} . Then, if $S_{i_u} \in C_t$ (with $t \leq i_u$), we have $T_{i_u} = x_{i_u} + d_{i_u} - d_t$, $u = 1, \dots, M$, and the total weighted tardiness of a candidate schedule is $Z = Z_m + \sum_{u=1}^M w_{i_u} T_{i_u} + \sum_{u=1}^M (\sum_{l=1}^{i_u} W_m^{i_{ul}}) x_{i_u}$. The algorithm outputs a schedule with minimum Z by tracing back the feasible transitions, starting from the state that has the Z_m which produced the minimum.

Theorem 1. *The dynamic programming algorithm above produces an optimal schedule.*

Note that in the proof of Theorem 1 we didn't need to check Conditions (1b),(2b). If, in addition, we require that the algorithm is non-preemptive, then the proof goes through without checking for Conditions (1c),(2c), since they are satisfied trivially by the optimal non-preemptive schedule:

Corollary 1. *The non-preemptive DP algorithm with feasible transitions restricted to only those that satisfy Conditions (1a), (2a) and (3) still produces an optimal (non-preemptive) schedule.*

Corollary 1 is used to establish the approximation ratio guarantee. We will compare the FPTAS-produced solution to the optimal schedule of the corollary.

5 The FPTAS

The transformation of the pseudopolynomial algorithm described in Sections 3, 4 into an FPTAS follows closely the FPTAS (Algorithm **Eps**) in [4]. Since the

running time of the dynamic programming part dominates the total running time, in what follows we use the term DP to refer to the entire process.

Let $\varepsilon > 0$. Recall that we have guessed Z^{ub} such that $Z^{ub}/2 \leq OPT \leq Z^{ub}$, and let $Z^{lb} := Z^{ub}/2$. Define $\delta = \frac{\varepsilon Z^{lb}}{4m}$. Consider a state $(k, Z_k^*, y_k^{i_1^*}, W_k^{i_1^*}, y_k^{i_2^*}, W_k^{i_2^*}, \dots, y_k^{i_M^*}, W_k^{i_M^*})$ of the exact dynamic programming. From this state, we will deduce the states $(k, Z_k, y_k^{i_1^1}, W_k^{i_1^1}, y_k^{i_2^1}, W_k^{i_2^1}, \dots, y_k^{i_M^K}, W_k^{i_M^K})$ used by the FPTAS dynamic programming as follows:

We round variable Z_k^* to the next multiple of δ (hence Z_k takes at most $\frac{Z^{ub}}{\delta} = O(\frac{n}{\varepsilon})$ distinct values). For every $1 \leq u \leq M$, we round $W_k^{i_u j^*}$ to the nearest power of $(1 + \varepsilon/2)^{1/m}$ (hence $W_k^{i_u j}$ takes $O(n \log W)$ values, where W is the total weight of the n jobs). After ordering the non-straddling jobs in WSPT order, let $w_{\pi(1)} > w_{\pi(2)} > \dots > w_{\pi(N)}$ be the $N \leq m$ distinct weight values of the m non-straddlers in decreasing order.

The rounding of $y_k^{i_u j^*}$, $1 \leq u \leq M$ is more complicated. Define a division of time interval $[0, \frac{Z^{ub}}{w_{\pi(N)}}]$ into subintervals $\{H_{i'} := [\frac{Z^{ub}}{w_{\pi(i'-1)}}, \frac{Z^{ub}}{w_{\pi(i')}}]\}_{i'=1}^N$.

In turn, divide each $H_{i'}$ into subintervals $\{\hat{H}_{i'j'}(i)\}_{j'=1}^{x_i^{i'}}$ of length $\delta_i = \frac{\delta}{w_{\pi(i')}}$

for all $1 \leq k \leq m$ and $1 \leq i \leq K$, where $x_i^{i'} = \lceil \frac{\frac{Z^{ub}}{w_{\pi(i')}} - \frac{Z^{ub}}{w_{\pi(i'-1)}}}{\delta_i} \rceil$ is the number of such subintervals (note that the length of the last subinterval may be less than δ_i). For each state $(k, Z_k, y_k^{i_1^1}, W_k^{i_1^1}, \dots, y_k^{i_M^K}, W_k^{i_M^K})$, the dynamic program applies its $O(K)$ transitions to generate new states $(k+1, Z_{k+1}, y_{k+1}^{i_1^1}, W_{k+1}^{i_1^1}, \dots, y_{k+1}^{i_M^K}, W_{k+1}^{i_M^K})$. For the set of states which have the same values of $Z_{k+1}, W_{k+1}^{i_1^1}, \dots, W_{k+1}^{i_M^K}$, we round $y_{k+1}^{i_u j}$ in the following way: we group all the $y_{k+1}^{i_u j}$ values that fall into the same subinterval $\hat{H}_{i'j'}$ together, and keep only the smallest and the largest values in this group, say $y_{k+1}^{i_u j^{max}}$ and $y_{k+1}^{i_u j^{min}}$. We emphasize that these two values correspond to the actual processing times of two sets of tardy jobs, and therefore none of these two values is greater than A_{k+1}^j . Hence, from the group of states generated by the DP transition, we produce and store states with at most two values at position y_{k+1}^{ij} , i.e., $(k+1, Z_{k+1}, y_{k+1}^{i_1^1}, W_{k+1}^{i_1^1}, \dots, y_{k+1}^{i_u j^{max}}, \dots, y_{k+1}^{i_M^K}, W_{k+1}^{i_M^K})$ and $(k+1, Z_{k+1}, y_{k+1}^{i_1^1}, W_{k+1}^{i_1^1}, \dots, y_{k+1}^{i_u j^{min}}, \dots, y_{k+1}^{i_M^K}, W_{k+1}^{i_M^K})$.

Lemma 8. *The algorithm runs in time $O((\varepsilon^{-1}n \log W \log P)^{\Theta(K^2)})$.*

We focus on states $(k, Z_k^*, y_k^{i_1^*}, W_k^{i_1^*}, \dots, y_k^{i_M^*}, W_k^{i_M^*})$, $k = 0, 1, \dots, m$ that are the sequence of transitions in the DP of Corollary 1 that produces an optimal non-preemptive schedule. The following lemma shows that despite the rounding used after every transition in our algorithm, there is a sequence of states $(k, Z_k, y_k^{i_1^1}, W_k^{i_1^1}, \dots, y_k^{i_M^K}, W_k^{i_M^K})$, $k = 0, 1, \dots, m$ whose transitions from one state to the next *match exactly* the job placement decisions of the optimal DP step-for-step. The key idea is that when our algorithm overestimates the space needed by tardy jobs (i.e., the y 's are rounded *up*), the space needed by the corresponding early jobs is decreased (rounded *down*), since the total space

needed remains the same, as (1) shows. The preemption of the tardy jobs allows us to treat the total space taken by the jobs in a class C_t as a unified entity, because the overestimated processing time of tardy jobs in this class can be placed (preempted) in the place of early jobs, whose processing time is reduced by an equal amount. This is the basic motivation behind our introduction of tardy job preemption.

Lemma 9. *For every $k = 1, 2, \dots, m$, given the identical placement of the first $k - 1$ jobs, if a placement of job J_k is feasible for the optimal DP, then the same placement is feasible for our DP.*

We work with these two special sequences and their transitions. We observe that $L_m^{0j*} \geq \sum_{h=1}^{u-1} p_{S_{i_h}} \forall j, u$ s.t. $1 < u \leq M$ and $i_{u-1} < j \leq i_u$ from Condition (3), which is satisfied by the optimal DP. Moreover, $L_m^{0l} \geq L_m^{0l*} \forall l$ s.t. $1 \leq l \leq K$ (cf. Lemma 9). Hence $L_m^{0j} \geq \sum_{h=1}^{u-1} p_{S_{i_h}} \forall j, u$ s.t. $1 < u \leq M$ and $i_{u-1} < j \leq i_u$, i.e., Condition (3) is satisfied by the last state produced by our algorithm in the sequence of transitions we study, and therefore we can feasibly complete the schedule produced in this way with the insertion of the tardy straddlers.

Theorem 2 proves the approximation ratio guarantee for the schedule produced by our algorithm, by proving this guarantee when the special transition sequence above is followed. We emphasize that our algorithm may not output the schedule corresponding to that sequence, since its approximate estimation of the total tardiness may lead it to picking another one, with a smaller estimate of the total tardiness. For every $1 \leq k \leq m$ and $1 \leq u \leq M$, let $B_{i_u}^*(k) := \max\{w_h \mid k \leq h \leq m, y_h^{i_u j} \neq 0, 1 \leq j \leq i_u\}$, and if no job is tardy in superinterval $G_{i_u i_{u+1}}$, set $B_{i_u}^*(k) := 0$. We can show that for every $1 \leq k \leq m$, $1 \leq u \leq M$, and $1 \leq j \leq i_u \leq K$, we have $Z_k \leq Z_k^* + 2k\delta$, and $0 \leq i_u B_{i_u}^*(k)(y_k^{i_u j} - y_k^{i_u j*}) \leq \delta$.

Theorem 2. *If Z is the total tardiness of the schedule returned by the algorithm and Z^* is the optimal, we have that $Z \leq (1 + \varepsilon)Z^*$.*

Till now we have assumed that each of the tardy straddlers straddles only one due date. It is easy to see how the algorithms can be modified to work for straddlers spanning more than one due date.

Acknowledgment. We thank George Steiner for enlightening discussions.

References

1. Abdul-Razaq, T.S., Potts, C.N., Van Wassenhove, L.N.: A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics* 26, 235–253 (1990)
2. Cheng, T.C.E., Ng, C.T., Yuan, J.J., Liu, Z.H.: Single machine scheduling to minimize total weighted tardiness. *Eur. J. Oper. Res.* 165, 423–443 (2005)
3. Du, J., Leung, J.Y.-T.: Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research* 15, 483–495 (1990)

4. Kellerer, H., Strusevich, V.A.: A Fully Polynomial Approximation Scheme for the Single Machine Weighted Total Tardiness Problem With a Common Due Date. *Theoretical Computer Science* 369, 230–238 (2006)
5. Kolliopoulos, S.G., Steiner, G.: Approximation Algorithms for Minimizing the Total Weighted Tardiness on a Single Machine. *Theoretical Computer Science* 355(3), 261–273 (2006)
6. Lawler, E.L.: A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Ann. Discrete Math.* 1, 331–342 (1977)
7. Lawler, E.L.: A fully polynomial approximation scheme for the total tardiness problem. *Operations Research Letters* 1, 207–208 (1982)
8. Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P.: Complexity of machine scheduling problems. *Ann. Discrete Math.* 1, 343–362 (1977)
9. McNaughton, R.: Scheduling with due dates and loss functions. *Management Sci.* 6, 1–12 (1959)
10. Sen, T., Sulek, J.M., Dileepan, P.: Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. *Int. J. Production Econom.* 83, 1–12 (2003)
11. Yuan, J.: The NP-hardness of the single machine common due date weighted tardiness problem. *Systems Sci. Math. Sci.* 5, 328–333 (1992)

On the Hardness and Approximability of Planar Biconnectivity Augmentation

Carsten Gutwenger, Petra Mutzel, and Bernd Zey

Department of Computer Science, Technische Universität Dortmund, Germany
`{carsten.gutwenger,petra.mutzel,bernd.zey}@tu-dortmund.de`

Abstract. Given a planar graph $G = (V, E)$, the *planar biconnectivity augmentation problem (PBA)* asks for an edge set $E' \subseteq V \times V$ such that $G + E'$ is planar and biconnected. This problem is known to be $\mathcal{NP}$ -hard in general; see [1]. We show that PBA is already $\mathcal{NP}$ -hard if all cutvertices of G belong to a common biconnected component B^* , and even remains $\mathcal{NP}$ -hard if the SPQR-tree of B^* (excluding Q-nodes) has a diameter of at most two. For the latter case, we present a new $5/3$ -approximation algorithm with runtime $\mathcal{O}(|V|^{2.5})$.

Though a $5/3$ -approximation of PBA has already been presented [2], we give a family of counter-examples showing that this algorithm cannot achieve an approximation ratio better than 2, thus the best known approximation ratio for PBA is 2.

1 Introduction

The problem of augmenting a graph to reach a certain connectivity has important applications in network design. For a fixed $k \in \mathbb{N}$, the *general augmentation problem* asks for a minimum number of edges to add to a graph such that the graph becomes k -connected. In this paper, we are only interested in the case $k = 2$, i.e., we want to add a minimum number of edges to obtain a biconnected graph. This problem can be solved in linear time [3]; however, if we consider planar graphs and demand that the augmented graph is still planar, the problem becomes $\mathcal{NP}$ -hard [1]. We call this problem the *planar biconnectivity augmentation problem (PBA)*. It has applications in graph drawing, where certain drawing algorithms require a planar and biconnected graph, e.g., [4]; if we want to apply such an algorithm to an arbitrary, not necessarily biconnected planar graph, we first need to augment the graph by adding edges. It is desirable to add as few edges as possible, since we do not want to change the graph too much.

Kant and Bodlaender [1] presented a rather simple 2-approximation algorithm for PBA that runs in $\mathcal{O}(n \log n)$ time for a graph with n vertices. They also show that the restricted case *PBA-Tric* where all cutvertices belong to the same triconnected component can be solved optimally in $\mathcal{O}(n^{2.5})$ time. Approximation algorithms for the general case with a better approximation factor than 2 have been proposed in the literature, but are all incorrect. Kant and Bodlaender [1] gave another approximation algorithm claiming this to be a $3/2$ -approximation, but Fialko and Mutzel [2] gave a counter-example showing that their algorithm

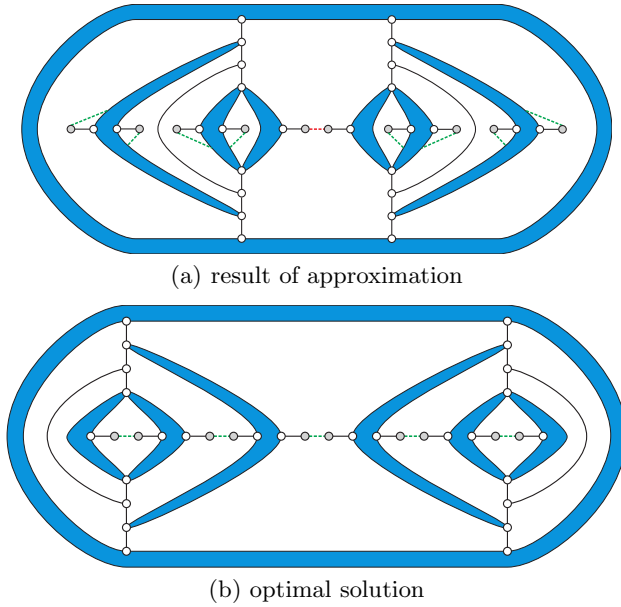


Fig. 1. Counter-example for $5/3$ -approximation with $k = 5$

has an approximation factor not better than 2. In the same paper, they proposed a new algorithm with approximation factor $5/3$. However, we show in the following that even this algorithm cannot have an approximation factor better than 2.

First, we need to introduce a few terms. Let G be a connected graph. The *BC-tree* $\mathcal{B}$ of G is the block-cutvertex tree of G , i.e., the relations between G 's biconnected components (*blocks*) and cutvertices. A leaf in $\mathcal{B}$ is called a *pendant*. A *bundle* B is a maximal set of pendants such that, for each pair $p, p' \in B$, $G + (p, p')$ is planar and adding (p, p') to G creates a new pendant in the corresponding BC-tree. In an augmenting edge set, an edge connecting two pendants from different bundles such that the number of pendants decreases by two is called *profitable* and we say the connected pendants are *cheap*. All other pendants are called *expensive*.

Our counter-example consists of a family of graphs G_k for odd $k \in \mathbb{N}$; Fig. 1 shows the graph G_5 as an example. Each graph consists of two identical subgraphs S_k , joined by a surrounding dense (i.e., triconnected) structure. Each S_k consists of alternating serial and parallel structures with $(k + 1)/2$ dense subgraphs, where the innermost such subgraph has one pendant and the other ones each have two pendants on opposite sides. Hence, G_k has in total $2k$ pendants.

The $5/3$ -approximation algorithm always chooses a largest bundle b_1 and then another largest bundle that can be connected with b_1 without losing planarity. In our case, all bundles have size 1, so the algorithm could connect the two pendants at the dense structures with only one pendant, as shown in Fig. 1(a).

But then, no other pair of pendants can be planarly connected anymore, and we end up with adding $2k - 1$ edges. On the other hand, in the optimal solution we only add k edges as shown in Fig. 1(b). Therefore, the approximation ratio is $2 - 1/k$. Since $\lim_{k \rightarrow \infty} (2 - 1/k) = 2$, we conclude that the approximation ratio of the algorithm cannot be better than 2.

Motivated by the facts that PBA is $\mathcal{NP}$ -hard and the best approximation known so far has a ratio of 2, but the restricted case PBA-Tric is optimally solvable in $\mathcal{O}(n^{2.5})$ time, we study the case *PBA-Bic* in which all cutvertices belong to a common biconnected component, called the *biconnected core*. In Sect. 3, we show that even a more restricted case, which we call *PBA-Bic** and formalize later, is already $\mathcal{NP}$ -hard. For this special case we give a new $5/3$ -approximation algorithm in Sect. 4 with $\mathcal{O}(|V|^{2.5})$ runtime. Though our new algorithm is only a $5/3$ -approximation for the special case PBA-Bic*, we hope that the ideas presented here will also be useful in solving more general cases.

2 Preliminaries

For a biconnected graph G , the *SPQR-tree* $\mathcal{T}$ of G represents its decomposition into triconnected components [5]. We only describe the idea of SPQR-trees briefly, please refer to [6] for a formal definition. The SPQR-tree $\mathcal{T}$ reflects the triconnectivity structure of G which is comprised of serial structures (*S-nodes*), parallel structures (*P-nodes*), and triconnected structures (*R-nodes*). With each node μ of $\mathcal{T}$, a *skeleton* graph G_μ is associated. According to the type of μ , its skeleton graph is either a cycle of at least three vertices (S-node), a bundle of at least three parallel edges (P-node), or a triconnected simple graph (R-node).

A skeleton can be seen as a sketch of G in the following sense. An edge (u, v) in G_μ is either a *real* edge corresponding to an edge (u, v) in G , or a *virtual* edge corresponding to a uv -component of G , i.e., a subgraph that is only attached at to vertices u and v to the rest of the graph. The respective uv -component of a virtual edge is determined by a neighbor η of μ whose skeleton G_η contains an edge (u, v) as well; η is also called the *pertinent node* of the virtual edge (u, v) . Hence, the skeletons of two adjacent nodes μ and η can be merged by contracting the edge (μ, η) , identifying the corresponding virtual edges in the skeletons, and finally removing the resulting virtual edge. Exhaustively merging skeletons recreates G .

We call the uv -component corresponding to a virtual edge e the *expansion graph* of e , and replacing a virtual edge e by its expansion graph is referred to as expanding e . If we consider $\mathcal{T}$ as a rooted tree, the *reference edge* of a node μ in $\mathcal{T}$ is the virtual edge e_{ref} in the skeleton graph G_μ whose pertinent node is the parent of μ , and the graph obtained by expanding all virtual edges in G_μ is the *pertinent graph* of μ .

Additionally, each edge of G can be represented by a Q-node whose skeleton is simply a 2-cycle (one virtual and one real edge), but we will not use Q-nodes here. The SPQR-tree of a graph can be constructed very efficiently, i.e., in linear time; see [7].

3 $\mathcal{NP}$ -Hardness of PBA-Bic*

Let G be a planar and connected graph. We define PBA-Bic* as the restricted version of PBA-Bic in which the SPQR-tree of G 's biconnected core (excluding Q-nodes) has a diameter of at most two. In this section, we proof that PBA-Bic* is NP-hard by constructing a polynomial-time reduction from the *planar vertex cover problem (PVC)* to the decision problem of PBA-Bic*. A *vertex cover* is a subset of vertices such that all edges have at least one endpoint in this set. PVC asks whether a planar and connected graph contains a vertex cover of cardinality k , or not. This problem is known to be $\mathcal{NP}$ -complete; see [8].

Let $\{G = (V, E), k\}$ be an instance of PVC and let $\Pi(G)$ be an arbitrary combinatorial embedding of G . We construct a graph $G' = (V', E')$ such that G has a vertex cover of size k if and only if G' can be augmented with $7.5q|E| - |V| + k$ edges, where $q := 4|V|$; compare Fig. 2.

In G' , each vertex $v \in V$ is represented by a planar, triconnected subgraph called the *vertex gadget* of v . This vertex gadget has two relevant faces f_v and f'_v separated by a triconnected subgraph called the *decision component*. The boundary of f_v is split up by $\deg(v)$ triconnected subgraphs called the *edge connection components*, inducing faces f_{vw} for all edges $(v, w) \in E$. The cyclic order of these components corresponds to the embedding $\Pi(G)$.

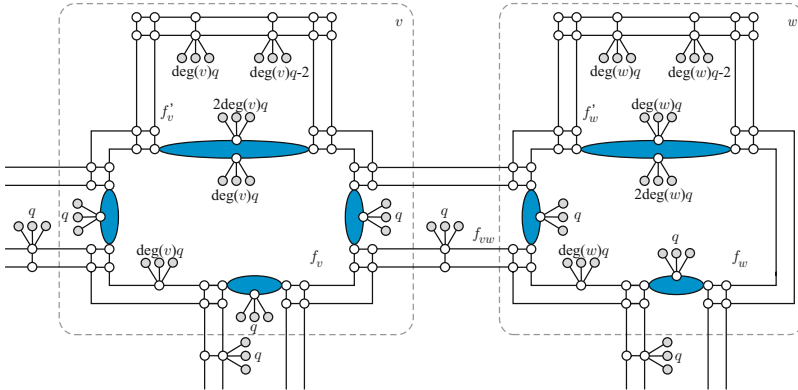


Fig. 2. Two adjacent vertex gadgets of the constructed graph G'

Furthermore, we add several bundles to G' . Each edge and edge connection component obtains one bundle with q pendants. Inside the two faces f_v and f'_v of v 's vertex gadget, there are in total $6 \deg(v)q - 2$ pendants: For the decision component, we add two bundles—one on each side—with $2 \deg(v)q$ and $\deg(v)q$ pendants, respectively. Furthermore, there are two bundles in face f'_v , one with $\deg(v)q$ and the other one with $\deg(v)q - 2$ pendants. And finally, a bundle with $\deg(v)q$ pendants is inserted into face f_v . Fig. 2 shows two adjacent vertex gadgets with $\deg(v) = 3$ and $\deg(w) = 2$, where the decision and the edge connection components are oriented contrarily.

Altogether, we have $7 \deg(v)q - 2$ pendants for each vertex $v \in V$ and q pendants for each edge in G . Hence, the total number of pendants is $\sum_{v \in V} (7 \deg(v)q - 2) + \sum_{e \in E} q = 15q|E| - 2|V|$.

Now we show that G has a vertex cover of size k if and only if G' can be augmented with $7.5q|E| - |V| + k$ edges. First, assume that G contains a vertex cover $V_{vc} \subseteq V$ of size k . By fixing the embedding of G' according to V_{vc} , G' can be augmented with $7.5q|E| - |V| + k$ edges: For every vertex $v \in V_{vc}$, the decision component of the corresponding vertex gadget is orientated such that the $2 \deg(v)m$ pendants are embedded into face f'_v . The pendants of each edge connection component are embedded into the faces representing the incident edges. For a vertex $w \notin V_{vc}$, the decision component is oriented the other way around and the q -bundles of all edge connection components of w are inserted into face f_w . Afterwards, each face of G' contains an even number of pendants and each pendant can be connected profitably, except for the pendants in the f'_v -faces with $v \in V_{vc}$. There, two pendants of the $2 \deg(v)q$ -bundle cannot be matched. Thus, G' can be augmented with $\frac{1}{2}(15q|E| - 2|V| - 2k) + 2k = 7.5q|E| - |V| + k$ edges.

Now assume that G' can be augmented with $7.5q|E| - |V| + k$ edges. Since the total number of pendants is $15q|E| - 2|V|$, it is easy to see that the augmentation contains exactly $2k$ expensive pendants. Since $q = 4|V| > 2k$, the only possible locations for expensive pendants are the f'_v -faces. We construct a vertex cover V_{vc} by adding a vertex $v \in V$ if and only if the corresponding vertex gadget in G' contains two expensive pendants. Since all q -bundles are cheap, every edge has at least one incident vertex in V_{vc} . Therefore, the set is a valid vertex cover of cardinality k .

Altogether, the construction of G' can be achieved in time $\mathcal{O}(|V| \cdot |E|)$, all cutvertices of G' belong to one biconnected component B^* , and the SPQR-tree of B^* (excluding Q-nodes) has a diameter of two. Since the decision problem of PBA-Bic* obviously belongs to $\mathcal{NP}$, the described transformation is a polynomial-time reduction from PVC to the decision problem of PBA-Bic* and the following theorem holds:

Theorem 1. *PBA-Bic* is $\mathcal{NP}$ -hard.*

4 A 5/3-Approximation for PBA-Bic*

In this section, we give a 5/3-approximation algorithm for PBA-Bic*. We will exploit the fact that one biconnected component of G , its biconnected core B^* , contains all cutvertices, thus allowing us to use the SPQR-tree data structure for representing all embeddings of B^* and estimating consequences of embedding decisions. The overall idea of the algorithm is to compute the SPQR-tree $\mathcal{T}$ of B^* (excluding Q-nodes), root it at the node with the maximum degree, extend the skeleton graphs with related pendants, and consider the tree nodes separately in a bottom-up traversal. After fixing the embedding for each expanded skeleton, a feasible and optimal augmentation can be computed easily; see [9] for details.

The algorithm proceeds as follows. Consider a node μ of $\mathcal{T}$ and the corresponding skeleton graph $G_\mu = (V_\mu, E_\mu)$. Since the skeletons only represent the triconnected structure, we extend the skeleton graphs with related pendants. We have two types of pendants: pendants whose corresponding cutvertices are vertices of G_μ , and pendants that are embedded into external faces of expansion graphs of virtual edges. The first set of pendants is simply added to the skeleton graphs, whereas the second set is represented by two values per edge, the *demand* for each side of the edge.

Since $\mathcal{T}$ has at most diameter two and we root it at a node with maximal degree, $\mathcal{T}$ contains only a root node and some leaf nodes. In case that μ is a leaf, each non-reference edge in G_μ is a real edge, hence every demand is zero and we can fix an arbitrary combinatorial embedding of G_μ . Afterwards, the pendants are embedded based on the computation of a maximum matching in an auxiliary graph $H = (V_H, E_H)$ constructed as follows. The vertices in V_H are in one-to-one correspondence to the pendants, and two vertices are connected if both corresponding pendants belong to different bundles and their cutvertices share a common face. Afterwards, each matched pair of pendants is embedded into this common face. Unmatched pendants which are adjacent to an external face count as demand for the corresponding skeleton edge in the parent node. In case of an S-skeleton, every unmatched pendant counts as demand on both sides.

If μ is the root of $\mathcal{T}$, we distinguish between its possible types:

- μ is an *R-node* or an *S-node*. Here, we have to determine the orientation of each virtual edge and the assignment of pendants to the adjacent faces. Thus, we select the edge with the maximum demand and orientate it such that the demand lies inside the face with the maximum number of remaining pendants. All involved edges and pendants are fixed and the procedure continues until all edges are embedded. Pendants whose cutvertices belong to the current skeleton are added to the face only if they would become cheap. This condition is only crucial for the largest bundle and can be checked easily. Finally free pendants are embedded using the same matching technique as for the leaves.
- μ is a *P-node*. For a P-skeleton, we have to determine the ordering of the parallel edges. We iteratively merge the two edges with the largest demands, i.e., they are fixed to be consecutive in the ordering and pendants are inserted into the face formed by the two edges, unless they would become expensive.

Theorem 2. *Let $G = (V, E)$ be a planar, connected graph, and let $\mathcal{S}'$ be an optimal solution of PBA-Bic* for G . Then, the algorithm described above augments G to a planar biconnected graph by adding at most $\frac{5}{3}|\mathcal{S}'|$ edges in $\mathcal{O}(|V|^{2.5})$ time.*

Proof (Sketch). Let $\mathcal{B}$ be the BC-tree of G and $\mathcal{P}$ the set of pendants in $\mathcal{B}$. Moreover, let $\mathcal{T}$ be the SPQR-tree of G 's biconnected core and $\mathcal{S}$ the solution computed by the algorithm.

We charge each pendant with costs $\frac{1}{2}$ or 1, depending on whether it is cheap or expensive in the solution. Thus, the costs of an added edge are partitioned

among the incident pendants, and the sum of all pendants' costs is the number of required edges for augmentation. Let $costs(\mathcal{P})$ and $costs'(\mathcal{P})$ denote the sum of the costs of the corresponding solutions $\mathcal{S}$ and $\mathcal{S}'$, respectively. We say a pendant is *bad* if it is expensive in $\mathcal{S}$ but cheap in $\mathcal{S}'$.

Each embedding decision causes some pendants to become cheap, while other pendants might become expensive because they cannot be connected profitably anymore. Though the embedding decisions are the reason for the occurrence of bad pendants, we say that the pendants of the current face are responsible for the affected pendants. Therefore, each pendant is virtually charged with the number of bad pendants it causes.

We show that the number of bad pendants can be compensated by the number of cheap ones, i.e., x cheap pendants cause at most $2x$ bad pendants. All other pendants $\mathcal{P}' \subseteq \mathcal{P}$ are expensive in $\mathcal{S}'$, and hence the costs of the constructed solution cannot be worse than the costs of the optimum solution for this set. Thus, the costs of $\mathcal{S}$ are at most $\frac{1}{2}x + 2x + costs'(\mathcal{P}')$ and the one of $\mathcal{S}'$ are at least $\frac{1}{2}x + x + costs'(\mathcal{P}')$. Therefore the ratio is at most $5/3$.

The proof is inductive over the sequence of embedding decisions and our induction hypothesis is as follows: Demanding pendants have not caused any bad pendants and bad pendants in the current face caused by previous decisions are already considered and can be compensated.

Base case: The base case occurs in a node μ of the SPQR-tree without any demanding pendants in the skeleton of μ . The algorithm computes a maximum matching M between the pendants of different bundles and embeds the pendants according to this matching. Therefore, each matched pendant can be connected profitably and, since the embedding decisions only concern the current skeleton, the number of affected pendants is bounded by the size of the matching. Hence, the $|M|$ cheap pendants cause at most $|M|$ bad pendants and the induction hypothesis is true. If the current node is also the root of $\mathcal{T}$, the augmentation is even an optimal solution.

Inductive step: Since $\mathcal{T}$ has only height one, we know that μ is the root node. We distinguish two cases:

μ is an R-Node (S-Node is similar): Let e be the currently selected edge with the maximal pendant set E_d , $e_d := |E_d|$, k the number of maximal possible pendants in an adjacent face, say f , of e , and f' the opposite face containing k' pendants. We will refer to the i -th virtual edge bordering f as e_i and the opposite face of e_i as f_i . The embedding decisions made by the algorithm are not critical with respect to the embedding of the whole graph, i.e., two pendants that belong to non-adjacent expansion graphs of the current face that can be connected before each decision are not affected.

Firstly, we will compute an upper bound of negatively affected pendants in other faces caused by the bounding pendants of f . Since each edge e_i is embedded such that the maximum demand k_i lies inside f , there are at most k_i affected pendants in f_i , for all i . In case edge e_i has already been fixed by a previous decision, the current embedding has no effect on that face. The same argument holds for the opposite face of e . However, in the unfixed case, f' contains at most

$k' \leq k$ demanding pendants, excluding the pendants attached to the other side of e , which are at most e_d . Hence, at most $\min\{k, e_d\}$ pendants are currently affected in expansion graphs adjacent to f' .

Secondly, we consider the current face f and the number of contained bad pendants. Since e_d is the maximum demand, $k_i \leq e_d$ holds for every i . The face f can be augmented optimally such that all expensive pendants belong to the set E_d . In case $k > e_d$ there are exactly $(k + e_d) \bmod 2 \leq 1$ expensive pendants. Otherwise, if $k \leq e_d$ holds, the number of expensive pendants is $e_d - k$. But since k is the maximal possible number, each expensive pendant either has been caused by previous decisions or cannot be bad. Hence, in case $k \leq e_d$, there are no new bad pendants.

Altogether, we have

$$k + \begin{cases} k & \text{if } k \leq e_d \\ e_d - ((k + e_d) \bmod 2) & \text{if } k > e_d \end{cases}$$

new cheap pendants and at most

$$k + \begin{cases} k & \text{if } k \leq e_d \\ e_d + ((k + e_d) \bmod 2) & \text{if } k > e_d \end{cases}$$

affected pendants. In case $k \leq e_d$ the ratio of cheap and bad pendants is at most 1, and the case $k > e_d$ leads to the worst-case ratio of the algorithm:

$$\frac{k + e_d - ((k + e_d) \bmod 2)}{k + e_d + ((k + e_d) \bmod 2)} \leq \frac{k + e_d - 1}{k + e_d + 1}$$

Since $e_d \geq 1$ and therefore $k \geq 2$ holds, the worst-case occurs if $e_d = 1$ and $k = 2$. Then, the number of bad pendants is two times the number of cheap pendants.

μ is a *P-Node*: The algorithm successively merges two virtual edges and adds pendants to the new face until they would become expensive. All other edges of $\text{skeleton}(\mu)$ can still be merged arbitrarily and the new edge can also be selected for following merge operations. Since all other demanding values are not larger than the second most demand, the same arguments hold as for the previous case.

The running time of the algorithm is dominated by the computation of a maximum matching inside the skeletons. Since the vertices of the auxiliary graph

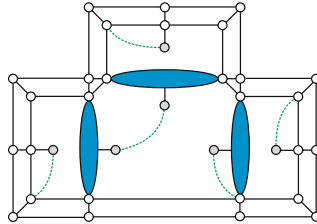


Fig. 3. A graph where the algorithm computes a solution with 5 edges (the dashed ones), whereas the optimum solution contains only 3 edges

G_H represent the pendants and edges are inserted between pendants of different bundles, the number of edges E_H may be $\Omega(|V_H|^2)$. Hence, a maximum matching can be computed in time $\mathcal{O}(|V_H|^{2.5})$ using the maximum matching algorithm by Micali and Vazirani [10], implying an upper bound of $\mathcal{O}(|V|^{2.5})$ for the whole algorithm. $\square$

The example in Fig. 3 shows that the analysis of the approximation ratio is also tight. Here, we have 6 pendants and our algorithm inserts 5 edges. However, by orientating the three triconnected subgraphs contrarily, 3 edges are sufficient to make the graph biconnected.

References

1. Kant, G., Bodlaender, H.L.: Planar graph augmentation problems. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1991. LNCS, vol. 519, pp. 286–298. Springer, Heidelberg (1991)
2. Fialko, S., Mutzel, P.: A new approximation algorithm for the planar augmentation problem. In: Proc. SODA 1998, SIAM, pp. 260–269. SIAM, Philadelphia (1998)
3. Hsu, T.S., Ramachandran, V.: On finding a smallest augmentation to biconnect a graph. *SIAM Journal on Computing* 22(5), 889–912 (1993)
4. Gutwenger, C., Mutzel, P.: Planar polyline drawings with good angular resolution. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 167–182. Springer, Heidelberg (1999)
5. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. *SIAM Journal on Computing* 2(3), 135–158 (1973)
6. Di Battista, G., Tamassia, R.: On-line planarity testing. *SIAM Journal on Computing* 25, 956–997 (1996)
7. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 77–90. Springer, Heidelberg (2001)
8. Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some simplified NP-complete graph problems. *Theoretical Computer Science* 1(3), 237–267 (1976)
9. Zey, B.: Algorithms for planar graph augmentation. Master's thesis, Dortmund University of Technology (2008), http://ls11-www.cs.uni-dortmund.de/people/gutweng/diploma_thesis_zey.pdf
10. Micali, S., Vazirani, V.V.: An $\mathcal{O}(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In: Proc. FOCS 1980, pp. 17–27. IEEE, Los Alamitos (1980)

Determination of Glycan Structure from Tandem Mass Spectra

Sebastian Böcker^{1,2}, Birte Kehr¹, and Florian Rasche¹

¹ Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, 07743 Jena,
Germany

{sebastian.boecker,florian.rasche}@uni-jena.de, BirteKehr@aol.com

² Jena Centre for Bioinformatics, Jena, Germany

Abstract. Glycans are molecules made from simple sugars that form complex tree structures. Glycans constitute one of the most important protein modifications, and identification of glycans remains a pressing problem in biology. Unfortunately, the structure of glycans is hard to predict from the genome sequence of an organism.

We consider the problem of deriving the topology of a glycan solely from tandem mass spectrometry data. We want to generate glycan tree candidates that sufficiently match the sample mass spectrum. Unfortunately, the resulting problem is known to be computationally hard. We present an efficient exact algorithm for this problem based on fixed-parameter algorithmics, that can process a spectrum in a matter of seconds. We also report some preliminary results of our method on experimental data. We show that our approach is fast enough in applications, and that we can reach very good *de novo* identification results. Finally, we show how to compute the number of glycan topologies of a given size.

1 Introduction

Glycans are – besides nucleic acids and proteins – the third major class of biopolymers, and are build from simple sugars. Since simple sugars can have up to four linkage sites, glycans are assembled in a tree-like structure. The elucidation of glycan structure remains one of the most challenging tasks in biochemistry, yet the proteomics field cannot be completely understood without these important post-translational modifications [1].

One of the most powerful tools for glycan structure elucidation is tandem mass spectrometry [5,15]. Mass spectrometry (MS) is a technology which, in essence, allows to determine the molecular mass of input molecules. Put in a simplified way, the input of the experiment is a molecular mixture and the output a peak list: a list of masses and their intensities. In tandem mass spectrometry, we select one type of molecules in the sample, fragment these parent molecules, and measure the masses of all fragments. Ideally, each peak should correspond to the mass of some sample molecule fragment, and its intensity to the frequency of that fragment in the mixture. The situation is, in fact, more blurred, due to noise and other factors. Tandem MS can provide general structural information

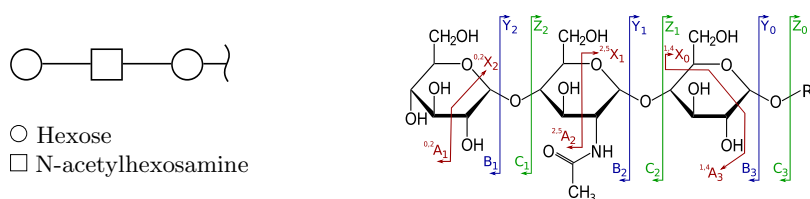


Fig. 1. Topology of a glycan made from three monosaccharides (left), and fragments resulting from tandem mass spectrometry analysis (right)

about the glycan, in particular its *topology* that can be represented as a labeled tree, see Fig. 1. Glycan mass spectra can be interpreted by searching a database of reference spectra [4], but such databases are vastly incomplete.

Recent approaches for *de novo* interpretation of tandem MS data usually build on two analysis steps, the first step being *candidate generation* (filtering) and the second step being *candidate evaluation*. A good candidate generation algorithm will generate a small set of candidates, but will not miss the correct interpretation. For glycans, a naïve approach to generate candidates is to decompose the parent mass of the glycan over the alphabet of monosaccharides [3], and then to enumerate all topologies that have the correct multiplicities of monosaccharides. Obviously, this is not feasible for large glycans.

Both candidate generation and candidate evaluation rely on certain scoring schemes, that are usually less sophisticated for candidate generation because of running time constraints: During candidate evaluation we only have to consider a small set of candidates. Recent approaches for tandem MS interpretation typically use scoring schemes that are elaborate modifications of the peak counting score, where one simply counts the number of peaks that are common to sample spectrum and candidate spectrum. Shan *et al.* [13] recently established that generating glycan topology candidates while avoiding peak double counts is an NP-hard problem. Existing approaches for glycan candidate generation can be subdivided into three categories: Some approaches enumerate all possible glycan topologies [7,8] and use strict biological rules to cut down on the number of candidates. Other tools use dynamic programming but simply ignore the problem of multiple peak counting [14]. Finally, Shan *et al.* [13] present a heuristic that avoids peak double counting.

Our contributions. We present a method that solves the candidate generation problem while at the same time, avoiding multiple peak counting. Although the corresponding problem is NP-hard, we present an exact method that allows to process a glycan tandem mass spectrum in a matter of seconds, and guarantees that all top-scoring candidate topologies are found. Our algorithm is fixed-parameter tractable [10] with respect to the parameter “number of peaks in the sample spectrum”. We report some preliminary results on experimental data, showing that our candidate generation performs well in practice. We also show that solving the simpler candidate generation problem where one allows multiple peak counting, usually leads to poor results. Additionally, we present a recurrence for counting glycan topologies of a given size.

2 Preliminaries

For glycans, collision-induced dissociation (CID) is often used as fragmentation technique in the tandem MS experiment. There are three types of fragmentation that break the glycan topology, resulting in six types of ions [6], see Fig. 1: X, Y, and Z-ions correspond to fragments that contain the monosaccharide attached to the peptide (the glycan’s reducing end) and are called *precursor ions* or *precursor fragments*. A, B, C-ions, in contrast, do not contain this monosaccharide. Using low collision energies in the fragmentation step, we predominantly generate B and Y ions, so we concentrate on these two types in our presentation.

Masses of molecules are measured in “Dalton” (Da), where 1 Dalton is approximately the mass of a neutron. We will often assume *integer masses* in our presentation for the sake of clarity. Accurate masses will be used in the scoring scheme.

We model a glycan topology as a rooted tree $T = (V, E)$, where the root is the monosaccharide attached to the peptide. Tree vertices are labeled with monosaccharides from a fixed alphabet Σ . Every vertex has an out-degree of at most four, because each monosaccharide has at most five linkages. Every element $g \in \Sigma$ and, hence, every vertex in the tree T is assigned an integer mass $\mu(g)$: this is the mass of the monosaccharide g , minus 18 Dalton for the mass of H_2O removed in binding. A fragment T' of T is a connected subtree, and the mass of T' is the sum of masses of the constituting vertices. Let $M := \mu(T)$ be the *parent mass* of the glycan structure. If we restrict ourselves to simple fragmentation events, then fragmentation of the tree means removing a single edge. Hence, we can represent each simple fragmentation event by a vertex $v \in V$, where the subtree $T(v)$ induced by v represents the non-precursor fragment, and the remainder of the tree is the precursor fragment. The resulting non-precursor fragments have the mass of a subtree of T induced by a vertex v , denoted $\mu(v)$. For precursor fragments we subtract $\mu(v)$ from the parent mass M . We ignore other mass modifications to simplify our presentation, such as the final H_2O , precursor fragment modifications through amino acids residues. These modifications can be easily incorporated into the presented methods.

Our method will take into account all possible glycan topologies, deliberately ignoring all biological restrictions. It is well known that certain branching types are observed seldom in biological samples: For example, most monosaccharides show only one to three linkages. But instead of completely forbidding such structures, we incorporate biological restrictions into our scoring model through penalties. In this way, we do not impede that rare structures can be found.

Our algorithm uses the concept of *fixed-parameter tractability* (FPT) [10]. This technique delivers exact solutions for an NP-hard problem in acceptable running time if the problem can be *parameterized*: In addition to the problem size n , we introduce a parameter k of the problem instance where typically $k \ll n$. A parameterized algorithm then restricts the exponential growth of its running time to the parameter k , whereas the running time is polynomial in n . Here, the problem size is the parent mass M whereas k is the number of (intense) peaks in the sample spectrum.

3 Candidate Generation

Assume we are given a candidate glycan tree T , and we want to evaluate T against the sample mass spectrum. We can use some simple fragmentation model to generate a hypothetical *candidate spectrum*, and use an additive scoring scheme to rate the candidate spectrum against the sample spectrum. Let $f(m)$ be the score we want to assign if a peak at mass m is present in our candidate spectrum. In its simplest incorporation, f is the characteristic function telling us if a peak is present in the *sample* mass spectrum at mass m . Then, summing $f(m)$ over all peak masses m in the candidate spectrum, we count all peaks that are common to both the sample spectrum and the candidate spectrum. We can also take into account expected peaks that are not present in the sample spectrum, by defining $f(m) = +1$ if a peak at mass m is present in the sample spectrum, and $f(m) = -1$ otherwise. Of course, for experimental data we use more involved scoring taking into account, say, peak intensities.

To simplify our presentation, let us assume for the moment that all our mass spectra consist of non-precursor ions only. Let $T = (V, E)$ be a labeled tree. Following Shan *et al.* [13] we define the scoring model $S'(T) := \sum_{v \in V} f(\mu(v))$. Unfortunately, scoring $S'(T)$ is not a peak counting score. Instead, for *every* subtree T' of T with mass $m' = \mu(T')$ we add $f(m')$ to the score: A tree that contains many subtrees of identical mass m' receives a high score if $f(m')$ is large even if it ignores all other peaks. We will show below how computations for this model can be transferred over to peak counting scores, though.

To find T that maximizes $S'(T)$, we define $S'[m]$ to be the maximal score of any labeled tree with total mass m . We use the simple recurrence from [13],

$$S'[m] = f(m) + \max_{m_1+m_2+m_3+m_4+\mu(g)=m} S'[m_1] + S'[m_2] + S'[m_3] + S'[m_4], \quad (1)$$

where the maximum is taken over all $g \in \Sigma$ and $0 \leq m_1 \leq m_2 \leq m_3 \leq m_4 \leq m$. We initialize $S'[0] = 0$. If one of the m_j in (1) equals zero, then the monosaccharide at the root of the subtree has less than four bonds. The maximal score of any glycan tree then is $S'[M]$, and we can backtrace through the array S' to find the optimal labeled tree. Shan *et al.* [13] simplify (1) to:

$$\begin{aligned} S'[m] &= f(m) + \max_{g \in \Sigma} \max_{m_1=0, \dots, \lfloor \frac{m-\mu(g)}{2} \rfloor} S'_2[m_1] + S'_2[m - \mu(g) - m_1] \\ S'_2[m] &= \max_{m_1=0, \dots, \lfloor \frac{m}{2} \rfloor} S'[m_1] + S'[m - m_1] \end{aligned} \quad (2)$$

The term $S'_2[m]$ corresponds to a “headless” subtree without a monosaccharide at its root. Using (2) we can compute $S'[M]$ in time $O(|\Sigma| \cdot M^2)$. Equations (1) and (2) can easily be modified to take into account properties of the monosaccharide g , such as the number of links of g for the scoring.

An exact algorithm for the peak counting problem. Recall that we actually want to compute the peak counting score $S(T) := \sum_{m=0, \dots, M} f(m) \cdot g(m)$ where $g(m)$ is the characteristic function of the labeled tree T : If T contains one

or more subtrees $T(v)$ with mass $\mu(v) = m$ then $g(m) = 1$, and $g(m) = 0$ if no such subtree exists. Unfortunately, finding the labeled tree T that maximizes $S(T)$ is an NP-hard problem [13].

We now modify recurrences (2) to find the labeled tree T that maximizes $S(T)$. To this end, note that the complexity of the problem only holds for mass spectra that contain a “large” number of peaks. But sample spectra are relatively sparse and contain only tens of peaks that have significant intensity: The number of simple fragments of a given glycan topology is only linear to its number of monosaccharides. Let k be the number of peaks in the sample spectrum: k is the parameter of our problem, and we limit the running time explosion to this parameter, while maintaining a polynomial running time with respect to M . For the moment, we assume that k is small; we will show below how to deal with mass spectra that contain a larger number of peaks.

In order to avoid multiple peak counting we incorporate the set of explained peaks into the dynamic programming. Scott *et al.* [12] introduced such an approach as part of their color-coding strategy. Let C^* be the set of peak masses in the sample spectrum, where $|C^*| = k$. For every mass $m \leq M$ and every subset $C \subseteq C^*$ we define $S[C, m]$ to be the maximal score of any labeled tree T with total mass $\mu(T) = m$ and only the peaks from C are used to compute this score. At the end of our computations, $S[C^*, M]$ holds the maximal score of any labeled tree where all peaks from C^* are taken into account for scoring. We now modify (2) for our purpose: We define $S_2[C, m]$ to be the score of a “headless” labeled tree with mass m using peaks in C . Using S_2 we can restrict the branching in the tree to bifurcations. We limit the recurrence of $S[C, m]$ to two subtrees with disjoint peak sets $C_1, C_2 \subseteq C$, where C_1 is the subset of peaks explained by the first subtree and C_2 is the set of peaks explained by the second subtree. We require $C_1 \cap C_2 = \emptyset$ what guarantees that every peak is scored only once. Additionally, we demand $C_1 \cup C_2 = C \setminus \{m\}$. Clearly, sets C that contain masses bigger than m need not be considered. We obtain the following recurrences:

$$\begin{aligned}
 S[C, m] &= \max_{g \in \Sigma} \max_{m_1=0, \dots, \lfloor \frac{m-\mu(g)}{2} \rfloor} \max_{C_1 \subseteq C \setminus \{m\}} \left\{ f(C, m) + S_2[C_1, m_1] \right. \\
 &\quad \left. + S_2[C \setminus (C_1 \cup \{m\}), m - \mu(g) - m_1] \right\} \quad (3) \\
 S_2[C, m] &= \max_{m_1=0, \dots, \lfloor \frac{m}{2} \rfloor} \max_{C_1 \subseteq C} S[C_1, m_1] + S[C \setminus C_1, m - m_1]
 \end{aligned}$$

Note that we delay the scoring of a peak at mass m if m is not in C by extending the scoring function to $f(C, m)$. If $m \notin C$ but $m \in C^*$ then $f(C, m) = 0$. Otherwise, set $f(C, m) = f(m)$. So, both peaks not in C^* and peaks in C are scored, whereas scoring of peaks in $C^* \setminus C$ is delayed.

We now analyze time and space requirements of recurrence (3). One can easily see that the space required to store $S[C, m]$ is $O(2^k \cdot M)$. Time complexity for calculating the optimal solution increases by a factor of 3^k reaching $O(3^k \cdot |\Sigma| \cdot M^2)$, as there are 3^k possibilities to partition k peaks into the three sets C_1 , C_2 ,

and $C^* \setminus (C_1 \cup C_2)$. The exponential running time factor can be reduced to 2^k [2], but the practical use seems to be limited due to the required overhead.

Recall that we have limited our computations to the case where only non-precursor ions are present in the mass spectra. We handle precursor ion by “folding” the spectrum onto itself, details will be given in the full paper.

To recover an optimal solution, we backtrack through the dynamic programming matrix starting from entry $S[C^*, M]$. Backtracking usually generates many isomorphic trees, which we remove from the final output, we defer the details. We can also compute all solutions that deviate at most δ from the score of the optimal solution, we omit the details. Running time of backtracking is $O(out \cdot 2^k \cdot Mn)$ where n is the maximal size of a glycan tree in the output, and out is the number of generated trees including isomorphic trees.

We note that several algorithm engineering techniques had to be used so that running times of our algorithm are as low as reported in Sec. 4. Due to space constraints, we defer all details to the full paper.

Scoring for candidate generation. The scoring presented above is overly simplified, what was done to ease the presentation. We now describe some modifications that are needed to achieve good results on real-world data. As noted above, our scoring has to be a simple additive scoring, and we only score fragments that stem from simple fragmentation events.

For our scoring we use real masses of fragments. All the presented recurrences iterate over integer masses m , but monosaccharide and subtree masses are non-integer. To deal with real masses, we define $S[m]$ to be the maximal score of any labeled tree whose exact mass falls into the interval $[m - 0.5, m + 0.5)$, and additionally store the exact mass of the subtree with optimal score in this interval. We update a matrix entry $S[m]$ only if the new subtree mass (the sum of $\mu(g)$, and masses of two headless subtrees) falls in the current interval. Note that for integer mass m_1 , we may have to consider the neighboring entries $\{m_1 - 1, m_1, m_1 + 1\}$ in the maximum (3), since the sum of corresponding exact masses might fall into the current interval.

Note that the optimal solution of this problem might no longer be the optimal solution of the original problem. But since we are interested in all solutions up to some δ away from optimality, chances are that the optimal solution of the original problem will be part of the set of candidates we generate. A similar reasoning applies if we limit exact computations to the k most intense peaks in the spectrum, because only peaks of low intensity and small score will be used multiple times.

The basis of our peak score is its normalized intensity, assuming that a high intensity indicates a high probability that the peak is not noise. Mass spectrometrists assume that the mass error of a device roughly is normal distributed. To account for mass deviation $\Delta = |m_{\text{peak}} - m_{\text{fragment}}|$, we multiply the peak intensity with $\text{erfc}(\Delta/(\sigma\sqrt{2}))$, where erfc is the error function complement and σ the standard deviation of the measurement error, typically set to $\frac{1}{3}$ or $\frac{1}{2}$ of the mass accuracy.

We do not apply a penalty if a peak in the sample spectrum is not explained by our candidate spectrum. This may be justified by the fact that our scoring ignores fragments not resulting from a simple fragmentation event. This scoring leads to good results as long as the intensity of peaks from simple fragmentation events is higher than that of non-simple ones.

4 Results on Experimental Data

We implemented our algorithm in Java 1.5. Running times were measured on an Intel Core 2 Duo, 2.5 GHz with 3 GB memory. A set of batroxobin carbohydrate side chains from Bothrops moojeni venom served us to test the program [9]. We used 24 spectra of N-glycans from recent investigations, where the compound was ionized by a single proton. The spectra were measured using a Bruker Daltonics ultraFlex TOF/TOF instrument with a MALDI ion source. Glycans are composed of fucoses (F, mass 146.06 Da), hexoses (H, 162.05 Da), and N-acetylhexosamines (N, 203.08 Da). Glycans were detached from the protein, and the reducing end was marked by a 2-aminopyridine modification resulting in a mass increase of 78 Da for precursor ions. The raw data was baseline corrected and peaks were picked using the SNAP method provided by Bruker. We use the naming convention from [9] for reporting the analyzed glycans.

We used the following parameters for analyzing the spectra: We set $k = 10$, avoiding multiple peak counting for the ten most intense peaks. We allowed a mass deviation of 1.0 Da and chose $\sigma = 0.5$ Da as standard deviation of the measurement error. After normalizing the sum of peak intensities we discarded all peaks with an intensity lower than 0.02. We chose the penalty for missing peaks as the average of the smallest intensity and the mean value of all peak intensities. We iteratively adjusted the score deviation δ for backtracking to obtain a candidate set of about 100 to 200 topologies. Results are shown in Table 1. In all but two cases (F4-4-H3N6F2, F6-1-H2N4F) the correct topology was part of this candidate set without any parameter tuning. We defer further details to the full paper. Average running time for generation of the candidates was 2.5 s without and 4.0 s including traceback.

To test if avoiding peak double-counting is needed for candidate generation, we set $k = 0$, so every peak could be counted an arbitrary number of times. Doing so, candidate generation produced the correct topology only for eight of the 24 spectra even if the candidate set was chosen to contain at least 500 structures. This shows that avoiding multiple peak counting is essential for the analysis. Certain glycan topologies do in fact create the same fragment mass several times: It must be understood that our approach does not *penalize* such topologies, but it also does not *reward* them.

Finally, we tested if further increasing k could improve the results of candidate generation. But as it turned out, increasing k to the 15 most intense peaks did not improve the results. So, computations can be carried out with a moderate k such as $k = 10$, without losing specificity.

Once we have reduced the set of potential glycan topologies from the exponential number of initial candidates, to a manageable set of tens or hundreds

Table 1. Results of the algorithm for 24 N-glycans of batroxobin [9]. Running times including traceback. *Due to some special properties of these two spectra, the parameter set had to be slightly changed to generate the correct candidate. See full version for details.

glycan	parent mass	# peaks	# B/Y ion peaks	δ	# cand.	running time	rank eval.
H3N5F	1744.0	36	10	15%	126	0.74 s	1
H3N6F	1947.0	41	8	10%	184	1.20 s	3
H3N6	1801.0	28	8	7%	122	0.93 s	1
H4N4F	1703.0	66	15	22%	115	4.69s	1
H5N4F	1865.0	34	11	15%	154	5.60 s	1
F1-H3N5	1598.6	26	7	25%	121	0.61 s	2
F2-1-H3N4F	1541.0	16	6	10%	133	0.43 s	2
F2-2-H4N4	1557.6	21	2	20%	128	1.75 s	1
F2-4-H4N4F	1703.6	33	12	20%	150	1.84 s	1
F2-5-H5N4F	1865.7	29	11	11%	282	0.92 s	1
F2-5-H5N4	1719.6	23	7	10%	147	1.09 s	2
F3-H3N8F	2354.0	18	6	12%	383	5.21 s	2
F4-1-H3N5	1598.0	35	12	20%	154	1.37 s	1
F4-3-H3N4F2	1687.6	20	6	17%	145	2.65 s	1
F4-3-H3N5F	1744.0	41	13	20%	113	3.03 s	1
F4-3-H4N2F2	1849.0	24	8	4.5%	164	5.40 s	4
F4-4-H3N6F2*	2092.0	11	8	9%	450*	4.04 s	2
F5-1-H3N6F	1947.0	92	11	6%	171	1.47 s	2
F5-1-H5N4F	1865.0	37	10	15%	169	3.14 s	3
F6-1-H2N4F*	1379.0	38	11	60%	103*	0.48 s	1
F6-2-H5N4F	1865.0	34	11	15%	161	6.48 s	1
F7-2-H4N4F	1703.0	66	15	23%	146	4.66 s	1
F7-3-H3N6	1801.0	28	8	7%	122	0.91 s	1
F7-3-H4N5F	1907.0	29	9	9%	143	0.87 s	2

of structures, we can now evaluate each candidate glycan topology using an in-depth comparison between its theoretical spectrum and the sample spectrum. This comparison can also take into account peculiarities of the mass spectrometry analysis, such as multiple-cleaved fragment trees, other ion series such as A/X and C/Z ions, or those X-ions that have lost parts of a monosaccharide. Evaluation of candidate glycan structures is not the focus of this work, we just report some identification rates we were able to obtain after evaluation. Our scoring generalizes ideas of Goldberg *et al.* [8], details are deferred to the full paper. The evaluation step ranked the true topology in all except four cases in the TOP 20, for 12 structures even on the first rank. In many cases, top-scoring topologies are biologically impossible. Using biological knowledge on the structure of the analyzed glycans, we always find the true solution in the TOP 4, and 14 true topologies reach rank one, see the right column of Table 1.

5 Number of Glycan Trees

We now show how to calculate the number $N[n, |\Sigma|]$ of different glycan topologies with n vertices, where vertices are labeled with elements from Σ . There exists a very rich and diverse treatment of similar tree counting problems in the literature, in areas such as mathematical chemistry or phylogenetics. To the best of our knowledge, neither the recurrence reported below, nor anything at least as fast has previously been reported in the literature.

Recall that a glycan topology corresponds to a rooted tree such that every vertex has out-degree at most four. For $|\Sigma| = 1$, Otter [11] analyzed the asymptotical behavior of this number which, in turn, allows us to approximate the number of glycan trees over an arbitrary alphabet Σ . In practice, this approximation is rather crude as we do not take into account isomorphic trees.

We now present a method for the exact computation of $N[n] := N[n, |\Sigma|]$, for alphabets Σ of arbitrary size. Due to space constraints, we only report the recurrence and leave all details and the formal proof of Lemma 1 to the full version of the paper. We claim

$$N[n+1] = |\Sigma| \cdot (N[n] + N_2[n, n] + N_3[n, n] + N_4[n, n]). \quad (4)$$

where $N[0] = 0$. The $N_i[n, k]$ for $i = 0, \dots, 4$ and $k \leq n_{\max}$ can be computed as

$$N_i[n, k] = \sum_{j=1}^i \sum_{m=\lceil n/j \rceil}^{\min\{k, \lfloor n/j \rfloor\}} \binom{N[m] + j - 1}{j} \cdot N_{i-j}[n - jm, m - 1] \quad (5)$$

We initialize $N_i[n, k]$ depending on i : For $i = 0$, we set $N_0[0, k] := 1$, and $N_0[n, k] := 0$ for all $n \geq 1$. For $i = 1$, we set $N_1[n, k] := N[n]$ for $n \leq k$, and $N_1[n, k] := 0$ otherwise. For $i \geq 2$, we set $N_i[n, k] := 0$ in case $i > n$ or $k \cdot i < n$ or $(k = 0 \text{ and } i > 0)$ holds. All other values can be computed from recurrences (4) and (5). For example, assume $|\Sigma| = 3$, then there exist $2.09 \cdot 10^7$ glycan trees with $n = 10$ vertices, and $2.15 \cdot 10^{18}$ glycan trees with $n = 30$ vertices. We have derived a similar recurrence for the number of glycan trees of a given mass m , we defer the details to the full paper.

Lemma 1. *Using recurrences (4) and (5), the number of glycan trees with n vertices over an alphabet Σ can be computed in time $O(n^3)$ and space $O(n^2)$.*

6 Conclusion

We have presented an approach for the automated analysis of glycan tandem mass spectra. We focused on the problem of candidate generation needed to reduce the search space of glycan structures. Despite the computational complexity of the candidate generation problem, our approach avoids peak double counting and solves the problem exactly using fixed-parameter techniques. We present a simple scoring scheme for candidate generation. Evaluation using experimental data shows that our method achieves swift running times and very good identification results.

Acknowledgments. We thank Kai Maaß from the biochemistry department of the Justus-Liebig-Universität Gießen for providing the glycan mass spectra.

References

1. Apweiler, R., Hermjakob, H., Sharon, N.: On the frequency of protein glycosylation, as deduced from analysis of the SWISS-PROT database. *Biochim. Biophys. Acta* 1473(1), 4–8 (1999)
2. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: *Proc. ACM Theor. Comp.*, pp. 67–74. ACM Press, New York (2007)
3. Böcker, S., Lipták, Z.: A fast and simple algorithm for the Money Changing Problem. *Algorithmica* 48(4), 413–432 (2007)
4. Cooper, C.A., Gasteiger, E., Packer, N.H.: GlycoMod – a software tool for determining glycosylation compositions from mass spectrometric data. *Proteomics* 1(2), 340–349 (2001)
5. Dell, A., Morris, H.R.: Glycoprotein structure determination by mass spectrometry. *Science* 291(5512), 2351–2356 (2001)
6. Domon, B., Costello, C.E.: A systematic nomenclature for carbohydrate fragmentations in FAB-MS/MS spectra of glycoconjugates. *Glycoconjugate J.* 5, 397–409 (1988)
7. Gaucher, S.P., Morrow, J., Leary, J.A.: STAT: a saccharide topology analysis tool used in combination with tandem mass spectrometry. *Anal. Chem.* 72(11), 2331–2336 (2000)
8. Goldberg, D., Bern, M., Li, B., Lebrilla, C.B.: Automatic determination of O-glycan structure from fragmentation spectra. *J. Proteome Res.* 5(6), 1429–1434 (2006)
9. Lochnit, G., Geyer, R.: Carbohydrate structure analysis of batroxobin, a thrombin-like serine protease from bothrops moojeni venom. *Eur. J. Biochem.* 228(3), 805–816 (1995)
10. Niedermeier, R.: *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, Oxford (2006)
11. Otter, R.: The number of trees. *The Annals of Mathematics* 49(3), 583–599 (1948)
12. Scott, J., Ideker, T., Karp, M., Sharan, R.: Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comput. Biol.* 13(2), 133–144 (2006)
13. Shan, B., Ma, B., Zhang, K., Lajoie, G.: Complexities and algorithms for glycan structure sequencing using tandem mass spectrometry. In: *Proc. of Asia Pacific Bioinformatics Conference (APBC 2007)*, *Advances in Bioinformatics and Computational Biology*, pp. 297–306. Imperial College Press (2007)
14. Tang, H., Mechref, Y., Novotny, M.V.: Automated interpretation of MS/MS spectra of oligosaccharides. *Bioinformatics* 21(suppl. 1), i431–i439 (2005); *Proc. of Intelligent Systems for Molecular Biology (ISMB 2005)*
15. Zaia, J.: Mass spectrometry of oligosaccharides. *Mass Spectrom. Rev.* 23(3), 161–227 (2004)

On the Generalised Character Compatibility Problem for Non-branching Character Trees

Ján Mañuch, Murray Patterson, and Arvind Gupta

School of Computing Science, Simon Fraser University, 8888 University Drive,
Burnaby, British Columbia, Canada

Abstract. In [3], the authors introduced the *Generalised Character Compatibility Problem* as a generalisation of the Perfect Phylogeny Problem for a set of species. This generalised problem takes into account the fact that while a species may not be expressing a certain trait, i.e., having teeth, its DNA may contain data for this trait in a non-functional region. The authors showed that the Generalised Character Compatibility Problem is NP-complete for an instance of the problem involving five states, where the characters' state transition trees are branching. They also presented a class of instances of the problem that is polynomial-time solvable. The authors posed an open problem about the complexity of this problem when no branching is allowed in the character trees. They answered this question in [2], where they showed that for an instance in which each character tree is $0 \rightarrow 1 \rightarrow 2$ (no branching), and only the states $\{1\}$, $\{0, 2\}$, $\{0, 1, 2\}$ are allowed, is NP-complete. This, however, does not provide an answer to the exact question posed in [3], which allows only one type of generalised state: $\{0, 2\}$, called here the Benham-Kannan-Warnow (BKW) Case.

In this paper, we study the complexity of various versions of this problem with non-branching character trees, depending on the set of states allowed, and depending on the restriction on the phylogeny tree: any tree, path or single-branch tree. In particular, we show that if the phylogeny tree is required to have only one branch: (a) the problem still remains NP-complete (for instance with states $\{1\}$, $\{0, 2\}$, $\{0, 1, 2\}$), and (b) the problem is polynomial-time solvable in the BKW Case (with states $\{0\}$, $\{1\}$, $\{2\}$, $\{0, 2\}$). We show the second result by unveiling a surprising connection to the Consecutive-Ones Property (C1P) Problem, used for instance, in DNA physical mapping, interval graph recognition and data retrieval.

1 Introduction

Constructing an evolutionary history from a set of species is the standard practice of computational evolutionary biologists [6]. A classical problem in this area is the *Perfect Phylogeny Problem*, the problem of trying to find a tree that explains the evolutionary relationship of a set of species. Here, each species is described by a set of *characters*. For a given species, a character only has a single

state (i.e., the number of legs is four, or the eye color is blue). The Perfect Phylogeny Problem is shown to be NP-complete in [4,13] and solvable in polynomial time when any of the associated parameters is fixed [1,8,9,11].

In [3], the authors argue why describing each species simply with characters that only have a single state for that species is not enough to accurately capture an evolutionary history from a set of species. In the human genome, only about 5% of the DNA is actually genes, the rest of it termed “junk” DNA, while some species have an even higher percentage of non-functional DNA. It can persist through evolution only because there is little or no selection pressure favoring streamlined genomes in multicelled organisms. Recent work has shown that this non-functional DNA can actually contain genes that were once active in the ancestors of a species, but are not presently expressed, perhaps due to mutations [7,10]. The most striking example of this is the demonstration that embryonic chicken tissue can be induced to differentiate into a tooth structure [10]. Although modern birds are well known not to have teeth, *Archaeopteryx*, their ancestral form of some 100 million years ago, was toothed.

Describing species with simple characters implies that a species only contains the data for the currently expressed state of the character. To remedy this, in [3], the authors propose the concept of *generalised characters*. Rather than having only a single state for a character and a species, a generalised character takes on a *set* of states (which we call generalised states, or just states when the context is clear), where we only know that the *expressed* state is in this set. This captures the fact that while a species is expressing a certain trait, it may contain the genetic information to express other traits as well. In addition to this, generalised characters also incorporate knowledge about the direction of transitions between character states (a transition from a double-sided mammalian heart to a hollow muscular tube heart will never happen, for example). The set of constraints on character state transitions is represented by a rooted tree, which we call a *character tree*, associated with the character. Here, the direction of character state transitions is always from parent to child in the character tree.

Since this new type of character is a generalisation of the “classical” character, there is also the notion of the Perfect Phylogeny Problem when species are described by generalised characters, which is termed the *Generalised Character Compatibility* (GCC) Problem, as opposed to the Classical Character Compatibility Problem. The authors of [3] present a polynomial-time algorithm for the case of the GCC Problem where for each character and each species, the set of states forms a directed path in the character tree. They also show that the general case is NP-complete. They do so by showing NP-completeness for a case when each character has five states, and the characters’ state transition trees are branching. However, in this setting the situation when a gene becomes hidden does not happen, therefore the authors posed an open problem about the complexity of the GCC Problem when the character trees are $0 \rightarrow 1 \rightarrow 2$ and the only states assigned to input species allowed are $\{0\}$, $\{1\}$, $\{2\}$ and $\{0, 2\}$ (we call this the Benham-Kannan-Warnow (BKW) Case). In [2], NP-completeness for a slight variant of this case in which we also allow the “wildcard” state $\{0, 1, 2\}$

(the information about the state for a particular species is unknown) was shown. We study the complexity of various variations of the BKW Case. In particular, we show that the GCC Problem when the character trees are not branching is NP-complete even when the phylogeny tree is required to have only one branch. We also show that the GCC Problem in the BKW Case can be solved in polynomial time when the phylogeny tree is required to have only one branch. Interestingly, the algorithm for this polynomial case is based on the algorithm for determining whether a 0-1-matrix has the Consecutive-Ones Property, used for instance, in DNA physical mapping, interval graph recognition and data retrieval [14].

The rest of this paper is organised as follows: In Sect. 2 we define the *Path Triple Consistency Problem* and show that it is NP-complete (a problem we use later in our proofs). We also restate in detail, the definition of the GCC Problem first defined in [3]. In Sect. 3 we show that the GCC Problem with non-branching character trees is NP-complete, even if the solution is required to be a single-branch tree. We then consider the BKW Case in Sect. 4, and show that it is polynomial-time solvable when the solution is required to be a single-branch tree. Finally, in Sect. 5, we conclude this paper with some new open problems.

2 Preliminaries

2.1 The Path Triple Consistency Problem

The Quartet Consistency (QC) Problem was shown by Steel [13] to be NP-complete. The QC Problem is, given a set $S = \{1, \dots, n\}$ and a set of quartets $\{a_i, b_i : c_i, d_i | i = 1, \dots, k\}$, where for $i = 1, \dots, k$, $a_i, b_i, c_i, d_i \in S$, is there a tree T on vertices S such that for each $i = 1, \dots, k$, there is an edge e_i of T whose removal separates vertices $\{a_i, b_i\}$ from vertices $\{c_i, d_i\}$. That is, the solution to the QC Problem is a tree where for every quartet, there is an edge separating that quartet. Surprisingly, if we restrict this tree to being a path, the problem is still NP-complete.

Since, in this restricted version of QC Problem, the quartets must all fall on a path, a quartet of the form $a, b : c, d$ can be replaced by the three constraints $a, b : c$, $a, b : d$ and $c, d : a$. Thus, we can re-pose this restricted version of the QC Problem as a set of triples of this form, where they must fall on a path. We will call this the Path Triple Consistency Problem. The formal definition of the problem is as follows:

Path Triple Consistency (PTC) Problem

Input: A set $S = \{1, \dots, n\}$ and a set of triples $\{a_i, b_i : c_i | i = 1, \dots, k\}$, where $a_i, b_i, c_i \in S$ for every $i = 1, \dots, k$.

Question: Is there a path (order) P on vertices S such that for each $i = 1, \dots, k$, there is an edge e_i of P whose removal separates vertices $\{a_i, b_i\}$ from vertex c_i .

Lemma 1. *The PTC Problem is NP-complete.*

Proof. The PTC Problem is actually complementary to the Total Ordering (TO) Problem, which was shown to be NP-hard by Opatrný in 1979 [12]. The TO Problem is, given a set $Q = \{1, \dots, n\}$ and a set of triples $\{a_i, b_i, c_i | i = 1, \dots, k\}$, where for $i = 1, \dots, k$, $a_i, b_i, c_i \in S$, is there a path (order) on Q such that for each $i = 1, \dots, k$, either $a_i < b_i < c_i$ or $c_i < b_i < a_i$. It is easy to see that the NP-completeness of the TO Problem implies the NP-completeness of the PTC Problem. Given instance of TO Problem $Q = \{1, \dots, n\}$ and $\{a_i, b_i, c_i | i = 1, \dots, k\}$, for the corresponding PTC Problem instance we let $S = Q$, and for each triple a, b, c of the TO Problem instance, we introduce the triples $a, b : c$ and $c, b : a$.

2.2 The Generalised Character Compatibility Problem

The Generalised Character Compatibility Problem [3] is a generalization of the Perfect Phylogeny Problem [4] where, instead of classical characters (where for a given species, a character only has one state for that species), the species set S is defined by a set of *generalised characters*. A generalised character is a pair $\hat{\alpha} = (\alpha, T_\alpha)$, such that:

1. α is a function $\alpha : S \rightarrow 2^{Q_\alpha}$, where Q_α denotes the set of states of $\hat{\alpha}$.
2. $T_\alpha = (V(T_\alpha), E)$ is a rooted character tree with nodes bijectively labelled by the elements of Q_α .

The Generalised Character Compatibility Problem is as follows:

Generalised Character Compatibility (GCC) Problem

Input: A set S of species and a set C of generalised characters.

Question: Is there a rooted tree $T = (V_T, E_T)$ and a “state-choosing” function $c : V_T \times C \rightarrow \bigcup_{\hat{\alpha} \in C} Q_\alpha$ such that the following holds:

1. For each species $s \in S$ there is a vertex v_s in T such that for each $\hat{\alpha} \in C$, $c(v_s, \hat{\alpha}) \in \alpha(s)$.
2. For every $\hat{\alpha} \in C$ and $i \in Q_\alpha$, the set $\{v \in T \mid c(v, \hat{\alpha}) = i\}$ is a connected component of T .
3. For every $\hat{\alpha} \in C$, the tree $T(\alpha)$ is an induced subtree of T_α , where $T(\alpha)$ is the tree obtained from T by labelling the nodes of T only with their α -states (as chosen by c), and then contracting edges having the same α -state at their endpoints.

Essentially, the first condition states that each species is represented somewhere in the tree T , and the second condition states that the set of nodes labelled by a given state of a given character form a connected subtree of T , just as with the Classical Character Compatibility Problem. Finally, condition three states that the state transitions for each character $\hat{\alpha}$ must respect its character tree T_α .

The GCC Problem is NP-complete. In particular, in [2] it was shown to be NP-complete for a case where for each species s and character $\hat{\alpha}$, $\alpha(s) \in$

$\{\{1\}, \{0, 2\}, \{0, 1, 2\}\}$, and T_α is $0 \rightarrow 1 \rightarrow 2$. It was also shown to be polynomial-time solvable in the case where for each species $s \in S$, $\alpha(s)$ is a directed path in T_α for each $\hat{\alpha} = (\alpha, T_\alpha) \in C$ [3]. We will consider the following variants of the GCC Problem. The GCC Problem with non-branching character trees (GCC-NB Problem) is a special case of the GCC Problem in which character trees have a single branch, i.e., each character tree T_α is $0 \rightarrow 1 \rightarrow \dots \rightarrow |T_\alpha| - 1$. If we restrict the solution of the GCC-NB Problem (a phylogeny tree) to have only one (or two) branches starting at the root, we will call this problem the Single-Branch GCC-NB Problem (SB-GCC-NB Problem), and the Path GCC-NB (P-GCC-NB Problem), respectively. In addition, if in any of these problems, say in problem X , we restrict the states to be from the set $\mathcal{Q}$, we will call this problem the $\mathcal{Q}$ - X Problem.

3 The SB-GCC-NB Problem Is NP-Complete

We first show that the GCC-NB Problem is NP-complete even when the solution is restricted to be a single-branch tree.

Theorem 1. *The $\{\{1\}, \{0, 2\}, \{0, 1, 2\}\}$ -SB-GCC-NB Problem is NP-complete.*

Proof. Given instance of the PTC Problem $S = \{1, \dots, n\}$, and the set of k triples $\{a_i, b_i : c_i\}$, we let S be the set of species, and let $C = \{\hat{\alpha}_1, \dots, \hat{\alpha}_k\}$ be the set of characters, where character $\hat{\alpha}_i$ corresponds to triple $a_i, b_i : c_i$. For $\hat{\alpha}_i \in C$, we let $\alpha_i(a_i) = \alpha_i(b_i) = \{1\}$ and $\alpha_i(c_i) = \{0, 2\}$, while for all other $s \in S \setminus \{a_i, b_i, c_i\}$ we let $\alpha_i(s) = \{0, 1, 2\}$.

Note that the constraints have been chosen in such a way that for each i , species c_i must be before both a_i and b_i , or after both a_i and b_i on the path, by choosing the i -th character of c_i to be 0 or 2, respectively. That is, the solutions to this instance of the SB-GCC-NB Problem correspond to solutions to the instance of the PTC Problem, and vice versa. The claim follows by Lemma 1.

Next, we show that if $\mathcal{Q} \subseteq 2^{\{0, \dots, m\}}$, the $\mathcal{Q}$ -SB-GCC-NB Problem is NP-complete, then $\mathcal{Q} \cup \{\{m\}\}$ -(P-)GCC-NB Problems are NP-complete.

Theorem 2. *If for $\mathcal{Q} \subseteq 2^{\{0, \dots, m\}}$, the $\mathcal{Q}$ -SB-GCC-NB Problem is NP-complete, then the $\mathcal{Q} \cup \{\{m\}\}$ -P-GCC-NB and $\mathcal{Q} \cup \{\{m\}\}$ -GCC-NB Problems are NP-complete.*

Proof. We will prove the claim by reduction from the $\mathcal{Q}$ -SB-GCC-NB Problem. An instance of the SB-GCC-NB Problem can be considered as an instance of the (P-)GCC-NB Problem, provided that we can force all species to be on a single branch. This can be done easily by adding the extra species x that has state set $\{m\}$ on all characters, and showing that all other species must have x as a descendant, which forces any solution to this instance of the (P-)GCC-NB Problem to be a single-branch tree. We omit the details.

As a corollary, we have that the $\{\{1\}, \{2\}, \{0, 2\}, \{0, 1, 2\}\}$ -(P-)GCC-NB Problem is NP-complete. However, the complexity of the BKW case posed in [3] remains open. In the next subsections, we will show that in the BKW case the SB-GCC-NB Problem is polynomial-time solvable.

4 The BKW Case of the SB-GCC-NB Problem Is Polynomial-Time Solvable

First, we show that the $\{\{1\}, \{0, 2\}\}$ -SB-GCC-NB and $\{\{1\}, \{0, 2\}\}$ -P-GCC-NB Problems are polynomial-time solvable, by showing that they are equivalent to the Consecutive-Ones Property Problem [5], used in DNA physical mapping, interval graph recognition and data retrieval. The formal definition of this problem is as follows:

Consecutive-Ones Property (C1P) Problem

Input: An $n \times m$ 0-1-matrix M .

Question: Is there an order on the m columns of M in such a way that, for any row, the set of columns that have entry 1 in that row are consecutive in the order.

This property can be determined in linear time by building a PQ -tree [5]. We then build on the algorithm for the C1P Problem (namely, the algorithm for building a PQ -tree) to show that the $\{\{0\}, \{1\}, \{2\}, \{0, 2\}\}$ -SB-GCC-NB Problem (the BKW Case of the SB-GCC-NB Problem) is also polynomial-time solvable.

Lemma 2. *The $\{\{1\}, \{0, 2\}\}$ -SB-GCC-NB and $\{\{1\}, \{0, 2\}\}$ -P-GCC-NB Problems are polynomial-time solvable.*

Proof. The solutions to the $\{\{1\}, \{0, 2\}\}$ -SB-GCC-NB and $\{1\}, \{0, 2\}$ -P-GCC-NB Problems must fall on a single-branch tree and path, respectively. Because $T_{\hat{\alpha}}$ is $0 \rightarrow 1 \rightarrow 2$ for any character $\hat{\alpha}$, all species where $\hat{\alpha}$ has state 1 must appear consecutively in this single-branch tree (path), otherwise there would be more than one transition from 0 to 1 in the phylogeny, for some character $\hat{\alpha}$. In this case of the SB-GCC-NB Problem, all other species can appear before (or after) this consecutive set of ones, because the “state-choosing” function c can map these species to 0 (or 2). As such, this problem is exactly the problem of determining whether or not a 0-1-matrix has the C1P, where each species is a column in this matrix. In this case of the P-GCC-NB Problem, if there does exist a solution P , then there is always a “state-choosing” function c' that reflects the fact that the corresponding matrix has the C1P. Therefore these cases are polynomial-time solvable.

We now consider the $\{\{0\}, \{1\}, \{2\}, \{0, 2\}\}$ -SB-GCC-NB Problem, the BKW case of the SB-GCC-NB Problem. Here, for any character $\hat{\alpha}$, a species s with $\alpha(s) = \{0, 2\}$ can still appear before or after the consecutive set of ones

(on this single-branch tree), however a species s with $\alpha(s) = 0$ has to appear *before* this set, while the species s with $\alpha(s) = 2$ has to appear *after* this set. So essentially, this is the problem of determining whether a 0-1-matrix has the C1P again, however the matrix, in addition to containing zeros and ones, contains some special zeros, we call them 0^- (0^+), that must appear before (after) the set of consecutive ones of its row, in any consecutive-ones ordering. This case is thus equivalent to the following generalised version of the C1P Problem:

Extended Consecutive Ones (E-C1P) Property Problem

Input: An $n \times m$ matrix M with entries 0, 1, 0^- or 0^+ .

Question: Is there an order on the m columns of M in such a way that, for any row, the set of columns that have entry 1 in that row are consecutive in the order, and any column that has entry 0^- (0^+) in that row appears before (after) this consecutive set of ones.

Lemma 3. *The E-C1P Problem is polynomial-time solvable.*

Proof. We prove this by showing that a structure that encodes all solutions to the E-C1P problem can be built in polynomial-time. Given instance M to the E-C1P Problem, we first construct PQ -tree PQ_M for matrix M , where we have “forgotten” the labels of the special zeros (we treat 0^- and 0^+ simply as 0). This can be done in $O(n+m)$ -time [5]. It is clear that PQ_M encodes a superset of the solutions to M . We then associate to each P node, the empty partial order on its children, and to each Q node, the set of directions $\{\text{left}, \text{right}\}$. Next, we obtain a list of order constraints imposed by the special zeros of M , by processing each pair of 0^- and 1, 0^+ and 1, and 0^- and 0^+ . For instance, if column i has 0^- and j has 1 in some row r , then we add constraint $i < j$ to the list. We now update these sets that are associated with each P and Q node, one-by-one from the list, to incorporate these ordering constraints. The idea is that these sets will restrict the configurations each node in PQ_M can have to the set of solutions of M .

When adding constraint $i < j$ from the list to PQ_M , we find the least common ancestor $a_{i,j}$ of i and j in PQ_M , which takes $O(m)$ steps:

- If $a_{i,j}$ is a Q node, then we eliminate from its set, the direction that places j before i . If the set of directions is now empty, then the algorithm halts, outputting that M does not have a solution. This can be done in constant time.
- Otherwise $a_{i,j}$ is a P node that stores some partial order on its children $\{v_1, \dots, v_k\} = V$. First, we find the children of $a_{i,j}$: v_x and v_y such that the subtrees rooted at them contain i and j , respectively. We add the constraint $v_x < v_y$ to the existing partial order at this P node. If this constraint is not consistent with the existing partial order then the algorithm halts, outputting that M does not have a solution. This partial order can be updated in time $O(k^2)$. Thus this step takes time $O(k^2) \subseteq O(m^2)$.

Since there are $O(nm^2)$ order constraints, and it takes time $O(m^2)$ to process each constraint, the algorithm takes time $O(nm^4)$.

Theorem 3. *The BKW case of the SB-GCC-NB Problem is polynomial.*

Proof. This follows from equivalence to the E-C1P Problem and Lemma 3.

5 Conclusions and Open Problems

We now conclude with a summary of results for the cases of the $\mathcal{Q}$ -GCC-NB Problem, $\mathcal{Q} \subseteq \{\{0\}, \{1\}, \{2\}, \{0, 2\}, \{0, 1, 2\}\}$, that are implied by our results, and what interesting cases are still open. This is conveyed in Fig. 1, where each row is a choice of $\mathcal{Q}$ for the $\mathcal{Q}$ -GCC-NB Problem, and the columns specify whether the solution is restricted to (a) a single-branch tree, (b) a path and (c) a tree, respectively. In Sect. 3, Theorem 1 shows that the GCC-NB Problem remains hard even when the solution is restricted to be a single-branch tree (8a in Fig. 1). Note that this also gives entries 1a, 6a and 7a. Theorem 2 then gives entries 1b,c and 7b,c. In Sect. 4, Lemma 2 gives entries 5a,b. Theorem 3 shows that the BKW Case of the SB-GCC-NB Problem is polynomial-time solvable, giving entries 2a, 3a and 4a. Note, that the BKW Case of the GCC-NB Problem, posed in [3] still remains open (2c in Fig. 1). The cases marked with (†) follow from [3,2]. In particular, notice that if $\mathcal{Q} \subseteq \{\{0\}, \{1\}, \{2\}, \{0, 1, 2\}\}$, that is each element in $\mathcal{Q}$ is a directed path in $0 \rightarrow 1 \rightarrow 2$, then the $\mathcal{Q}$ -GCC-NB Problem is polynomial-time solvable by [3]. Entries 13a,b,c involve only classical characters, and are thus polynomial-time solvable by [1]. The entries marked with (Δ) follow from our new preliminary results not included in this paper. In all remaining cases the problem becomes trivial as there always is a solution.

$\mathcal{Q} \backslash \text{soln}$	(a) branch	(b) path	(c) tree
(1) $\{\{0\}, \{1\}, \{2\}, \{0, 2\}, \{0, 1, 2\}\}$	NP-c	NP-c	NP-c †
(2) $\{\{0\}, \{1\}, \{2\}, \{0, 2\}\}^*$	P	NP-c Δ	?
(3) $\{\{0\}, \{1\}, \{0, 2\}\}$	P	NP-c Δ	?
(4) $\{\{1\}, \{2\}, \{0, 2\}\}$	P	?	?
(5) $\{\{1\}, \{0, 2\}\}$	P	P	?
(6) $\{\{0\}, \{1\}, \{0, 2\}, \{0, 1, 2\}\}$	NP-c	NP-c Δ	NP-c †
(7) $\{\{1\}, \{2\}, \{0, 2\}, \{0, 1, 2\}\}$	NP-c	NP-c	NP-c †
(8) $\{\{1\}, \{0, 2\}, \{0, 1, 2\}\}$	NP-c	?	NP-c †
(9) $\{\{0\}, \{2\}, \{0, 2\}\} \cup \{\{0, 1, 2\}\}$	P Δ	NP-c Δ	P †
(10) $\{\{0\}, \{1\}, \{0, 1, 2\}\} \cup \{\{2\}\}$	P Δ	NP-c Δ	P †
(11) $\{\{0\}, \{2\}, \{0, 1, 2\}\}$	P Δ	NP-c Δ	P †
(12) $\{\{1\}, \{2\}, \{0, 1, 2\}\}$	P Δ	NP-c Δ	P †
(13) $\mathcal{Q} \subseteq \{\{0\}, \{1\}, \{2\}\}$	P	P	P †

Fig. 1. Complexity of various cases of the GCC-NB Problem (*) BKW Case (†) implied by [3,2] (Δ) new preliminary results

We now give comments on some of the cases that remain open. The $\{\{1\}, \{0, 2\}\}$ -SB-GCC-NB Problem (the solution must be a single-branch tree) corresponds naturally to the C1P Problem (the solution must be a total order). As such, the $\{\{1\}, \{0, 2\}\}$ -GCC-NB Problem corresponds to a tree-C1P Problem which is defined as follows: Find a tree with vertex set containing the columns of a binary matrix M and auxiliary columns such that for every row in M' , a

tree labelled by the row contracts to a tree $0 - 1 - 0$ after contracting any two vertices with the same label, where M' is a matrix containing M and auxiliary columns. Similarly, one could define the tree-E-C1P Problem on matrices with 0^- , 0 , 0^+ and 1. Now, determining the complexity of this tree-E-C1P Problem would provide an answer to the BKW Case of the GCC-NB Problem [3].

References

1. Agarwala, R., Fernandez-Baca, D.: A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM J. on Computing* 26(6), 1216–1224 (1994)
2. Benham, C.J., Kannan, S., Paterson, M., Warnow, T.: Hen's teeth and whale's feet: Generalized characters and their compatibility. *Journal of Computational Biology* 2(4), 515–525 (1995)
3. Benham, C.J., Kannan, S., Warnow, T.: Of chicken teeth and mouse eyes, or generalized character compatibility. In: Galil, Z., Ukkonen, E. (eds.) *CPM 1995*. LNCS, vol. 937, pp. 17–26. Springer, Heidelberg (1995)
4. Bodlaender, H., Fellows, M., Warnow, T.: Two strikes against perfect phylogeny. In: Kuich, W. (ed.) *ICALP 1992*. LNCS, vol. 623, pp. 273–283. Springer, Heidelberg (1992)
5. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property of, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and Systems Sciences* 13(3), 335–379 (1976)
6. Felsenstein, J.: Numerical methods for inferring evolutionary trees. *The Quarterly Review of Biology* 57(4), 379–404 (1982)
7. Janis, C.: The sabertooth's repeat performances. *Natural History* 103, 78–82 (1994)
8. Kannan, S., Warnow, T.: Inferring evolutionary history from DNA sequences. *SIAM J. on Computing* 23(4), 713–737 (1994)
9. Kannan, S., Warnow, T.: A fast algorithm for the computation and enumeration of perfect phylogenies. In: *SODA*, pp. 595–603 (1995)
10. Kollar, E.J., Fisher, C.: Tooth induction in chick epithelium: Expression of quiescent genes for enamel synthesis. *Science* 207, 993–995 (1980)
11. McMorris, F.R., Warnow, T., Wimer, T.: Triangulating vertex colored graphs. *SIAM J. on Discrete Mathematics* 7(2), 296–306 (1994)
12. Opatrny, J.: Total ordering problem. *SIAM J. on Computing* 8(1), 111–114 (1979)
13. Steel, M.: The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification* 9, 91–116 (1992)
14. Telles, G.P., Meidanis, J.: Building PQR trees in almost-linear time. In: *Proc. of GRACO*, pp. 33–39 (2005)

Inferring Peptide Composition from Molecular Formulas

Sebastian Böcker^{1,2} and Anton Pervukhin¹

¹ Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany
sebastian.boecker@uni-jena.de, apervukh@minet.uni-jena.de

² Jena Centre for Bioinformatics, Jena, Germany

Abstract. With the advent of novel mass spectrometry techniques such as Orbitrap MS, it is possible to determine the exact molecular formula of an unknown molecule solely from its isotope pattern. But for protein mass spectrometry, one is facing the problem that many peptides have exactly the same molecular formula even when ignoring the order of amino acids. In this work, we present an efficient method to determine the amino acid composition of an unknown peptide solely from its molecular formula. Our solution is based on efficiently enumerating all solutions of the multi-dimensional equality constrained integer knapsack problem.

1 Introduction

Novel mass spectrometry techniques allow us to determine the mass of a sample molecule with very high accuracy of 5 ppm (parts-per-million), and sometimes below 1 ppm [10,9]. These techniques are increasingly coupled with high throughput separation techniques such as (Ultra) High Performance Liquid Chromatography, and have become one preferred method for the analysis of peptides [8] and metabolites [14]. In proteomics, this is of particular interest to detect post-translational modifications [13], or non-ribosomal peptides that are not directly encoded in the genome [2]. With the advent of new MS instruments such as Orbitraps, mass spectra with very high mass accuracy will be routinely acquired for protein identification and quantification in the near future.

It has been known for almost two decades that one can infer the molecular formula of a sample molecule solely from its isotope pattern [7, 12]. But only recently, measurement accuracy has increased to a point where this analysis is feasible for sample molecules with mass of 1000 Dalton and above [5]. In addition, efficient methods had to be developed to carry out the computational analysis for larger molecules in reasonable running time [5,6]. The upper mass limit of molecules that allow for this interpretation, is ever increasing due to improvements in existing MS techniques as well as the development of new ones.

Given an isotope pattern of an unknown peptide, one can decompose its monoisotopic mass [6] and then score amino acid decompositions with regards to their theoretical isotope pattern [5]. Clearly, this is a non-trivial problem since there exist about $3.96 \cdot 10^{11}$ amino acid decompositions with mass up to

2500 Dalton. Unfortunately, many peptides have exactly the same molecular formula even when ignoring the order of amino acids: Besides leucine and isoleucine, the smallest non-trivial example are peptides consisting of two glycine vs. a single asparagine, both with molecular formula $C_4H_8N_2O_3$. Using the above technique, we repeatedly score amino acid decompositions with identical molecular formula and, hence, identical isotope pattern. On the other hand, the sample may be contaminated by metabolite molecules that have a molecular formula which cannot be explained by any peptide. By determining the molecular formula for the isotope pattern, we can effectively sort out such contaminants. So, it is much more efficient to first determine the molecular formula of a sample from its isotope pattern, and then to compute all amino acid compositions that match the molecular formula.

Our contributions. Our input is the molecular formula of an unknown peptide. Our goal then is to find all amino acid compositions that match the given molecular formula. We formulate the problem as a joint decomposition of a set of queries or, equivalently, as a multi-dimensional equality constrained integer knapsack problem [1]. Our queries are the number of carbon, hydrogen, and other atoms that make up the molecule. We present the dimension reduction method that reduces a multi-dimensional problem to a one-dimensional decomposition problem, which in turn can be efficiently solved using methods presented in [6]. We also provide an experimental evaluation of the algorithm’s running time, both on simulated data and peptides from experimental mass spectra. We find that our *mixed matrix* approach is the fastest method for enumerating solutions, and is one to two orders of magnitude faster than the runner-up algorithm.

2 Preliminaries

Proteins and peptides are made up from the five elements hydrogen (symbol H), carbon (C), nitrogen (N), oxygen (O), and sulfur (S). For each amino acid we know its exact molecular formula, such as $C_3H_7N_1O_2$ for alanine. When an amino acid is added to a peptide chain, it loses a water molecule. In the following, we concentrate on amino acid *residues* that are missing a water molecule H_2O , so an alanine residue has molecular formula $C_3H_5N_1O_1$. Note that leucine and isoleucine have identical molecular formula and cannot be told apart by mass spectrometry. In the following, we treat these two amino acids as one, and talk about 19 standard amino acids.

In this paper, we want to find all amino acid compositions that match a given molecular formula. To approach this problem, we can use branch-and-bound search by adding amino acids as long as for each element, the resulting molecule contains at most as many atoms as the input molecule, and output exact hits. Alternatively, we can compute the molecular formulas of all amino acid compositions up to a certain mass during preprocessing, and use hashing to efficiently search this list. Particularly the latter approach suffers from the large number of amino acid decompositions, see above.

We want to approach the problem of decomposing a molecular formula over the amino acid alphabet, as a multi-dimensional equality constrained integer knapsack problem. Recall that we ignore isoleucine in our presentation. Now, we can formulate our problem as a matrix multiplication $Ax = b$ where A is a matrix containing multiplicities of all elements in amino acids, x is the 19-dimensional vector we search for, and b is the input molecular formula over the elements CHNOS. But one can use one more trick to further simplify the problem: only amino acids methionine and cysteine contain sulfur. So, if our input molecular formula contains k sulfur atoms, we first try to distribute these between methionine and cysteine, iterating over all possibilities $M_0C_k, M_1C_{k-1}, \dots, M_kC_0$. In each case, we reduce the input molecular formula accordingly, and we try to decompose the resulting formulas over the remaining 17 amino acids A, D, E, F, G, H, K, L, N, P, Q, R, S, T, V, W, Y:

$$A := \begin{pmatrix} 3 & 4 & 5 & 9 & 2 & 6 & 6 & 6 & 4 & 5 & 5 & 6 & 3 & 4 & 5 & 11 & 9 \\ 5 & 5 & 7 & 9 & 3 & 7 & 12 & 11 & 6 & 7 & 8 & 12 & 5 & 7 & 9 & 10 & 9 \\ 1 & 1 & 1 & 1 & 1 & 3 & 2 & 1 & 2 & 1 & 2 & 4 & 1 & 1 & 1 & 2 & 1 \\ 1 & 3 & 3 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 1 & 2 & 2 & 1 & 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} \# \text{ C} \\ \# \text{ H} \\ \# \text{ N} \\ \# \text{ O} \end{pmatrix}, \quad Ax = b \quad (1)$$

Here, x is a 17-dimensional vector representing the remaining amino acid residues.

3 Single-Dimensional Integer Knapsack

We first consider the single-dimensional equality constrained integer knapsack [1]

$$a_1x_1 + a_2x_2 + \dots + a_nx_n = b \quad (2)$$

where a_j are integer-valued coefficients usually satisfying $a_j \geq 0$, and $b \geq 0$. We search for all solution vectors $x = (x_1, \dots, x_n)$ such that all x_j are non-negative integers. We start with an important observation: if there exist indices i, j with $a_i > 0$ and $a_j < 0$, and if (2) has at least one solution, then there is an *infinite* number of solutions. In the following, we assume $a_j \geq 0$ for all j .

One can use dynamic programming to efficiently compute all solutions of (2) [6]: We choose a maximal integer B that we want to decompose, and construct a bit table of size $n \times B$ during preprocessing. Using this table, we can efficiently find all solutions (2) for all queries $b \leq B$. An alternative method for finding all solutions uses a Extended Residue Table of size $n \cdot a_1$, see [6] for details. Here, every decomposition is constructed in time $O(na_1)$ independent of the input b . In addition, we do not have to choose a maximal integer B during preprocessing. This latter method also appears to be faster in practice. Finally, we can count the exact number of decompositions $\gamma(b)$ of the integer b using a dynamic recurrence similar to the bit table mentioned above, see again [6].

The number of decompositions over coprime integers $a_1, \dots, a_n$ asymptotically behaves like a polynomial of degree $n - 1$ in b [15]:

$$\gamma(b) \sim \frac{1}{(n-1)! a_1 \dots a_n} b^{n-1}. \quad (3)$$

We can use this formula or the more precise version from [3], to approximate the number of amino acid decompositions $\hat{\gamma}(M, \epsilon)$ with *real mass* in the interval $[M, M + \epsilon]$, over the 19 standard amino acids:

$$\hat{\gamma}(M, \epsilon) \approx 1.12687 \cdot 10^{-55} \epsilon M^{18} + 2.29513 \cdot 10^{-51} \epsilon M^{17} + 2.16611 \cdot 10^{-47} \epsilon M^{16}$$

Unfortunately, this approximation is very inaccurate for masses below 10 000 Da. Due to space constraints, we defer a better approximation to the full paper.

4 Multi-dimensional Integer Knapsack

We now generalize (2) to the multi-dimensional equality constrained integer knapsack problem: We want to find all solutions of the equation $Ax = b$ for $A = (a_{i,j})_{1 \leq i \leq d, 1 \leq j \leq n}$ where $a_{i,j}$ are integer-valued coefficients satisfying $a_{i,j} \geq 0$, and $b_i \geq 0$. We search for all solution vectors $x = (x_1, \dots, x_n)$ such that all x_j are non-negative integers. This corresponds to d one-dimensional knapsack equations (2) that we want to solve simultaneously, and it is a special case of a Diophantine equation, where all entries are non-negative.

A simple algorithm to compute all solutions of $Ax = b$, is to choose one row $i \leq d$ as the master row, then to find all solutions of the one-dimensional integer knapsack $a_{i,1}x_1 + \dots + a_{i,n}x_n = b_i$ and, finally, to test for each solution of the master equation if the solution also satisfies the other rows of matrix A . We call this the *naïve decomposition* algorithm. However, this involves generating many decompositions unnecessarily. Decompositions of the master equation can be found by recursing through the Extended Residue table, see [4]. Böcker *et al.* [4] also present a method that can be seen as an intermediate between the naïve decomposition algorithm, and the method presented below: The *multiple decomposition* algorithm also chooses a master equation to decompose, but tests during recursion if all other equations of $Ax = b$ besides the master equation can still be satisfied using the current partial solution. If this is no longer possible, then it stops and discards the current partial solution.

Equation (3) tells us that the number of solutions increases with a polynomial of degree $n - 1$. So, it seems advisable to lower n as much as possible, if we can do so. In fact, the multi-dimensional knapsack gives us an opportunity to lower n : To this end, we apply a Gaussian elimination to matrix A to find a lower triangular matrix $L \in \mathbb{R}^{d \times d}$ and an upper triangular matrix $R = (r_{i,j}) \in \mathbb{R}^{d \times n}$ such that $A = LR$. Then, $Ax = b$ if and only if $Rx = L^{-1}b =: b'$ where L^{-1} is known. Every solution of $Ax = b$ must hence satisfy the *bottom equation* of R ,

$$0 \cdot x_1 + \dots + 0 \cdot x_{d-1} + r_{d,d}x_d + \dots + r_{d,n}x_n = b'_d \quad (4)$$

that has at most $n - d + 1$ non-zero coefficients. We now search for all solutions of the bottom equation, and we test for each one if it is also a solution of $Ax = b$.

If all entries in A are integers, we can easily guarantee the same to be true for the output matrix R . But it should be understood that even if $a_{i,j} \geq 0$ holds for all coefficients, we cannot guarantee $r_{i,j} \geq 0$ for all coefficients after

Gaussian elimination. In particular, there may be negative coefficients in the bottom equation. But as we have learned in Sec. 3, this implies that there is an infinite number of solutions for the bottom equation, provided that there exists at least one solution. Hence, we have to avoid that negative coefficients appear in the bottom equation. Can we find Gaussian eliminations schemes where all coefficients of the bottom equation are non-negative? To do so, we may have to permute the columns and rows of A : We choose a permutation π of the rows of A , and a permutation σ of the columns of A that brings d columns to the front but ignores the remaining $n - d$ columns. We have $d!$ possibilities to choose π , and $(n - d + 1) \cdots n$ possibilities to choose d front rows of A in σ .

We use the following simple version of the Gaussian elimination algorithm in our computations: Assume that rows and columns have been swapped in matrix A . Set $\tilde{L} := L^{-1}$, we will compute $\tilde{L}$ instead of L . We initialize $\tilde{L} = (l_{i,j}) \leftarrow I$ as the identity matrix and $R \leftarrow A$. We iterate the following for $i = 1, \dots, d - 1$. For rows $i' = i + 1, \dots, d$ and columns $j = i, \dots, n$ we define the new submatrix $r'_{i',j} = r_{i',i}r_{i,j} - r_{i,i}r_{i',j}$. Then, $r'_{i',i} = 0$ must hold. Similarly, for rows $i' = i + 1, \dots, d$ and columns $j' = 1, \dots, n$ we compute the new submatrix $l'_{i',j'} = r_{i',i}l_{i,j'} - r_{i,i}l'_{i',j'}$. But if $r_{i,i} = 0$ this operation reduces the rank of our matrix R , and the resulting matrix is no longer equivalent to our input matrix. In consequence, we STOP if we encounter the case $r_{i,i} = 0$.

In case we do not drop out off the elimination algorithm, we test if all entries of the bottom equation are non-positive: In this case, we negate the bottom equation. Finally, we check if $r_{d,j} \geq 0$ holds for all $j = d, \dots, n$: Otherwise, we have to discard R, L^{-1} . Different permutations π might lead to the same bottom equation if σ is kept constant, so we finally have to sort out duplicate bottom equations. We end up with a list of elimination matrix pairs R, L^{-1} that all allow us to compute decompositions for the multi-dimensional problem using their bottom equations: Assume that for one such pair R, L^{-1} we are given a input vector b . First, we apply the row permutation π to b , that was used to generate A . Then, we find all solutions of the equation $Rx = L^{-1}b$ as follows: We compute $b' \leftarrow L^{-1}b$, and we use one of the decomposition techniques for single-dimensional integer knapsacks on the bottom equation of $Rx = b'$. For every decomposition $(x_d, \dots, x_n)$, we iterate $i = d - 1, d - 2, \dots, 1$ and compute entry x_i using row i of $Rx = b'$. We test if $x_i \geq 0$ and if x_i is integer; otherwise, we discard the decomposition. Finally, we apply the inverse column permutation σ^{-1} to x . Doing so for all decompositions of the bottom equation, guarantees that we find all solutions of $Ax = b$. On the other hand, many decompositions can be generated “in vain” because these are no solutions of $Ax = b$.

5 Mixed Matrix Approach

We have used Gaussian elimination for all row and column permutations, a total of $1 \cdot 2 \cdot 3 \cdot 4 \cdot 14 \cdot 15 \cdot 16 \cdot 17 = 1370880$ possibilities. In 43176 cases, the reduction scheme generated a bottom equation with non-negative entries. After discarding identical bottom equations, we ended up with only 19 matrix pairs R, L^{-1} .

Now, one question remains: Which of these matrix pairs R, L^{-1} is “the best”? Different matrix pairs usually differ in the number of decompositions that are generated in vain, and that have to be discarded. If we use efficient decomposition techniques from [6] for single-dimensional decomposition, then we can guarantee that running time for generating decompositions is actually linear in the number of decompositions. Then, the number of discarded decompositions is an excellent indicator for the quality of a matrix pair.

In our evaluations we use the notion of *competitive ratio*: the ratio between the number of true decompositions, over the number of decompositions generated by a particular matrix pair. Using a training set of molecular formulas to decompose, we can filter out matrices that generate too many additional candidates. To find the exact number of decompositions of a particular matrix, we can use the dynamic programming techniques mentioned in Sec. 3.

Although being an ideal indicator for evaluation purposes, in application one would like to avoid the explicit calculation of the number of decompositions, because this can be very time consuming. Therefore, the following question inevitably arises: Can we estimate the number of discarded decompositions, without actually calculating decompositions? To this end, set $b'_d := \sum_{k=1}^d l_{d,k} b_k$ as the number we actually want to decompose in the bottom equation (4). Recall that the number of decompositions of b over n coprime integers asymptotically behaves like a polynomial of degree $n - 1$, see (3). So, we define

$$\tilde{l}(b) := \frac{1}{(m-1)! r_{n-m+1,d} \cdots r_{n,d}} \left(\sum_{k=1}^d l_{d,k} b_k \right)^{m-1} \quad (5)$$

as our *indicator*, where m is a number of non-zero values in the bottom row of R .

Our *mixed matrix* approach now works as follows: Given a vector b to decompose, we compute the indicators $\tilde{l}(b)$ for every matrix pair L^{-1}, R and choose the matrix pair with the smallest indicator. Then, we use the bottom equation of the corresponding matrix R to actually decompose b , see the previous section.

6 Experimental Results

To evaluate our approach we proceed as follows: First, we calculate the competitive ratios of all matrices and filter out those that generate too many candidates in vain. The number of decompositions is not the only factor affecting the running time. In addition, the time required to filter out incorrect solutions may vary for different matrix pairs. To better estimate the actual running time for a particular matrix pair, we will apply a slight correction to our indicator. We also compare running times of the mixed matrix approach to several other algorithms.

Datasets. For our evaluations we use two datasets. The first dataset with 6000 peptides has been simulated by in-silico digestion (trypsin) using the Swiss-Prot database (release 56.5), eliminating duplicates. We select peptides with masses between 900 and 1500 Da, and for each mass range of 100 Da we randomly choose 1000 peptides. The second dataset consists of 99 peptides from *de novo*

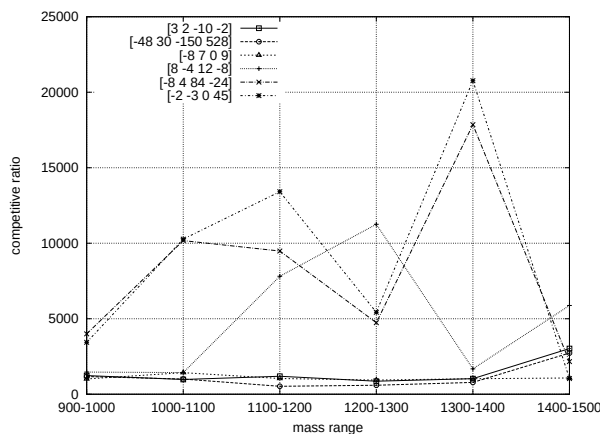


Fig. 1. Competitive ratios of six matrices with the lowest competitive ratios. Average competitive ratio for mass bins of width 100 Da.

interpretations of experimental mass spectra, acquired on quadrupole ion-trap mass spectrometer. These peptides range in mass from about 900 to 2000 Da.

Choosing Good Matrix Pairs. For each matrix pair we calculate the competitive ratio for all peptides in the simulated dataset, so that we can filter out matrix pairs that generate too many false positives. We calculate the average competitive ratio over all peptides for bins of size 100 Da. In Fig. 1 we have depicted the competitive ratios of the six best matrices. Matrices are labeled by the last row of the matrix L^{-1} . One can see that three matrices show outstanding competitive ratios for all mass ranges. For the remaining 13 matrices, we find that the average competitive ratio never drops below 3900 for any matrix and any mass bin of size 100 Da (data not shown).

We selected the three matrix pairs with the best competitive ratios for further evaluation. For each matrix pair we calculate the indicator $\tilde{l}(b)$ and compare it with the actual running time for the input vector b . We observe an almost linear correlation between logarithms of indicators and running times, data not shown. We also observe a slight shift of the intercept of the linear fit over the running time for various matrices. This corresponds to the differences in running times required for filtering false decompositions. We derive an indicator correction from this experimental data, we omit the details.

To find the matrix pair that we actually use in our *mixed matrix* approach to decompose b , we apply the linear correction to the indicator $\tilde{l}(b)$ and choose the matrix with the minimal value. Clearly, this is not necessarily the matrix pair with minimal running times. We want to evaluate how often we choose a suboptimal matrix pair from these three pairs: For the simulated dataset, we choose the correct matrix pair in more than 97.5% of the cases, resulting in an overall running time increase of less than 0.4%. Results on the real data are

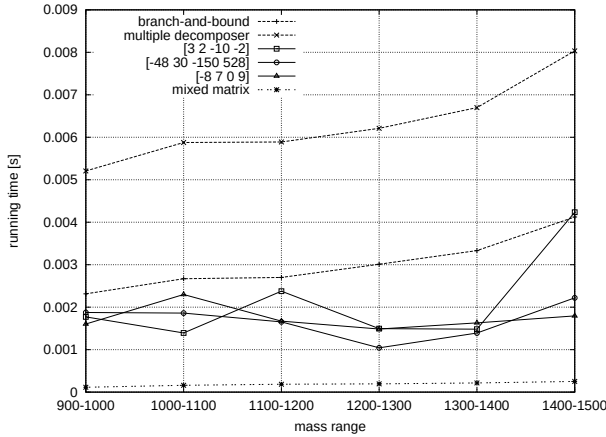


Fig. 2. Running times of the algorithms (in seconds) for simulated data. Running times are calculated per decomposition and averaged over all peptides in the mass range. We also report the performance of our method with only one particular matrix pair applied.

similar: in less than 5 % of the cases a suboptimal matrix is chosen, resulting in a total running time increase of 0.8 %.

Comparison with Other Methods. Finally, we want to evaluate how good the mixed matrix method works compared to branch-and-bound searching, the naïve decomposition algorithm [6], and the multiple decomposition algorithm [4]. In all cases, we distribute sulfur atoms between methionine and cysteine.

The branch-and-bound search first tries all possibilities for alanine and branches, then does the same for aspartic acid, and so on until we reach the last amino acid tyrosine. The naïve decomposition algorithm simply uses one of the rows of matrix A to compute decompositions, and then test if any such decomposition satisfies $Ax = b$. Both for this and the multiple decomposition algorithm, we use Extended Residue Table to compute decompositions [6]. In fact, there exist four different flavors of the latter two algorithms, as we can choose one of the rows of matrix A from (1) as our master row. For these two methods, we only report the best results of the four possibilities.

Fig. 2 and 3 show running times of these approaches on simulated and real data. Running times for the naïve decomposition algorithm are significantly worse than those of all other approaches and, hence, omitted. For both datasets our mixed matrix approach significantly outperforms the second best approach, branch-and-bound searching. We observe a 16-fold speedup over the branch-and-bound algorithm, and a 35-fold speedup over the multiple decomposition algorithm on average. Running times of the mixed matrix approach range from 0.11 to 0.25 milliseconds per decomposition, measured for the simulated dataset. On the real dataset, speedup of the mixed matrix approach reaches 67-fold over the branch-and-bound algorithm. We also observe that the mixed matrix approach significantly outperforms each individual matrix pairs, see Fig. 2.

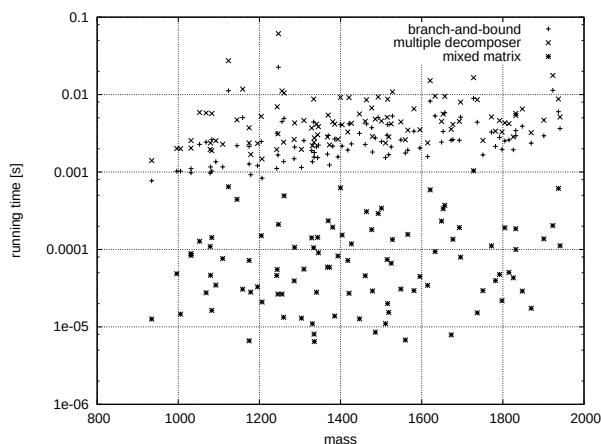


Fig. 3. Running times of the algorithms (in seconds) for real data

We have also evaluated matrix pairs with five rows, that include sulfur as a decomposable element. Performance for these matrix pairs was in all cases significantly slower than what we have presented above.

All algorithms were implemented in C++, and running times measured on an AMD Opteron-275 2.2 GHz with 6 GB of memory running Solaris 10.

7 Conclusion

We have presented an efficient method to enumerate all solutions of a multi-dimensional equality constrained integer knapsack problem. We have applied our method to the problem of finding all amino acid compositions with a given molecular formula. First results on both simulated and real data show the outstanding performance of our *mixed matrix* approach which is one to two orders of magnitude faster than the runner-up method. We are currently conducting further comparisons of our algorithm with other related methods such as [11], that solve linear Diophantine systems with negative coefficients.

We can easily include more matrix pairs for computing decomposition, what seems advisable for molecular formulas with mass above 1400 Da. We can speed up the decomposition process by eliminating duplicates in the bottom row of R , distributing the resulting number between the “merged” amino acids. Finally, note that the first row of R contains only positive entries, resulting in upper bounds for amino acids that can be dynamically updated during backtracking.

Clearly, our method can be used for any application where we have to enumerate all solutions of a multi-dimensional equality constrained integer knapsack [1]. This is necessary whenever finding an optimal solution of the knapsack cannot be modeled via a simple linear or quadratic objective function. For example, the mixed matrix method can be used to speed up the search for a molecular formula of an unknown sample molecule, as proposed in [4].

Acknowledgments. AP supported by Deutsche Forschungsgemeinschaft (BO 1910/1). We thank Andreas Bertsch from the Division for Simulation of Biological Systems of the Tübingen University for providing the peptide dataset.

References

1. Aardal, K., Lenstra, A.K.: Hard equality constrained integer knapsacks. In: Cook, W.J., Schulz, A.S. (eds.) IPCO 2002. LNCS, vol. 2337, pp. 350–366. Springer, Heidelberg (2002)
2. Bandeira, N., Ng, J., Meluzzi, D., Linington, R.G., Dorrestein, P., Pevzner, P.A.: De novo sequencing of nonribosomal peptides. In: Vingron, M., Wong, L. (eds.) RECOMB 2008. LNCS (LNBI), vol. 4955, pp. 181–195. Springer, Heidelberg (2008)
3. Beck, M., Gessel, I.M., Komatsu, T.: The polynomial part of a restricted partition function related to the Frobenius problem. *Electron. J. Comb.* 8(1), N7 (2001)
4. Böcker, S., Letzel, M., Lipták, Z., Pervukhin, A.: Decomposing metabolomic isotope patterns. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 12–23. Springer, Heidelberg (2006)
5. Böcker, S., Letzel, M., Lipták, Z., Pervukhin, A.: SIRIUS: Decomposing isotope patterns for metabolite identification. *Bioinformatics* 25(2), 218–224 (2009)
6. Böcker, S., Lipták, Z.: A fast and simple algorithm for the Money Changing Problem. *Algorithmica* 48(4), 413–432 (2007)
7. Fürst, A., Clerc, J.-T., Pretsch, E.: A computer program for the computation of the molecular formula. *Chemom. Intell. Lab. Syst.* 5, 329–334 (1989)
8. Haas, W., Faherty, B.K., Gerber, S.A., Elias, J.E., Beausoleil, S.A., Bakalarski, C.E., Li, X., Ville, J., Gygi, S.P.: Optimization and use of peptide mass measurement accuracy in shotgun proteomics. *Mol. Cell. Proteomics* 5(7), 1326–1337 (2006)
9. He, F., Hendrickson, C.L., Marshall, A.G.: Baseline mass resolution of peptide isobars: A record for molecular mass resolution. *Anal. Chem.* 73(3), 647–650 (2001)
10. Olsen, J.V., de Godoy, L.M.F., Li, G., Macek, B., Mortensen, P., Pesch, R., Makarov, A., Lange, O., Horning, S., Mann, M.: Parts per million mass accuracy on an orbitrap mass spectrometer via lock mass injection into a c-trap. *Mol. Cell. Proteomics* 4, 2010–2021 (2006)
11. Papp, D., Vizvári, B.: Effective solution of linear diophantine equation systems with an application in chemistry. *J. Math. Chem.* 39(1), 15–31 (2006)
12. Rockwood, A.L., Van Orden, S.L.: Ultrahigh-speed calculation of isotope distributions. *Anal. Chem.* 68, 2027–2030 (1996)
13. Tanner, S., Payne, S.H., Dasari, S., Shen, Z., Wilmarth, P.A., David, L.L., Loomis, W.F., Briggs, S.P., Bafna, V.: Accurate annotation of peptide modifications through unrestrictive database search. *J. Proteome Res.* 7, 170–181 (2008)
14. von Roepenack-Lahaye, E., Degenkolb, T., Zerjeski, M., Franz, M., Roth, U., Wessjohann, L., Schmidt, J., Scheel, D., Clemens, S.: Profiling of Arabidopsis secondary metabolites by capillary liquid chromatography coupled to electrospray ionization quadrupole time-of-flight mass spectrometry. *Plant Physiol.* 134(2), 548–559 (2004)
15. Wilf, H.: *Generating functionology*, 2nd edn. Academic Press, London (1994)

Optimal Transitions for Targeted Protein Quantification: Best Conditioned Submatrix Selection

Rastislav Šrámek¹, Bernd Fischer², Elias Vicari¹, and Peter Widmayer¹

¹ Department of Computer Science, ETH Zurich, Zurich, Switzerland
`{rsramek,vicari,el,widmayer}@inf.ethz.ch`

² European Bioinformatics Institute, Cambridge, United Kingdom
`bfischer@ebi.ac.uk`

Abstract. Multiple reaction monitoring (MRM) is a mass spectrometric method to quantify a specified set of proteins. In this paper, we identify a problem at the core of MRM peptide quantification accuracy. In mathematical terms, the problem is to find for a given matrix a submatrix with best condition number. We show this problem to be NP-hard, and we propose a greedy heuristic. Our numerical experiments show this heuristic to be orders of magnitude better than currently used methods.

Keywords: bioinformatics, submatrix selection problem, minimal condition number.

1 Introduction

This paper discusses the algorithmic problem of finding a best conditioned submatrix of a given matrix, as it results from the biological measurement problem of multiple reaction monitoring (MRM) in tandem mass spectrometry. MRM is a very promising method for measuring the quantity (abundance) of proteins in biological samples [1,2,3,4,5]. For details about the biological problem that motivates our study, consult the technical report [6].

Mass spectra for peptides are represented by the number of ions one can detect with the mass spectrometer in a certain state. The state of the spectrometer is governed by three main parameters: retention time (t), first mass to charge value ($ms1$) and second mass to charge value ($ms2$). We call this triple of parameters a *transition*. In a preliminary step, spectra on transitions are harvested for different peptides. In the actual quantification experiment, the measured spectra will be the sum of the spectra of the peptides in the mixture, their quantities taken into account. Our goal is to select very few transitions that will be measured. We want to do this in a way that will allow us to estimate the composition of the sample with greatest expected accuracy.

We suppose that for every peptide, the set of transitions ($t, ms1, ms2$) is contained in a database together with the previously observed ion count ic . A natural way of looking at the database of transitions and ion counts for each

peptide is a $n \times m$ matrix T , where n is the number of all possible transitions and m is the number of different peptides. The entry $t(i, j)$ then denotes the ion count peptide j has at transition i . Obviously, $n > m$ for any sensible database. Let x be a vector of all peptide abundances. Since there are multiple peptides that share the same transition, the observed ion count for transition i is the scalar product of $T(i, \cdot)x$. Let c be the vector of observed ion counts of all transitions (observed by multiple reaction monitoring), then the peptide abundances can be estimated by solving the overdetermined least squares problem

$$\hat{x} = \min_x \|Tx - c\|. \quad (1)$$

For this solution to be plausible, we would have to assume that we can measure the ion counts c of all n transitions, which is infeasible due to hardware and time constraints. We need to carefully choose some transitions to measure and discard the other data.

Perhaps the most intuitive way to tackle the transition selection problem mentioned before is to select as many transitions as different peptides, construct a fully determined system of m linear equations with m variables and solve it. The selection of transitions in this case amounts to selecting the corresponding rows of the matrix T to form a $m \times m$ submatrix S that is the matrix representation of the left side of the system of linear equations we will be solving.

The transitions should be selected in such a way that the information we acquire on the peptide abundances x is maximized. A good candidate for the optimization function is the condition number of this matrix. The condition number of a matrix A in a system of linear equations $Ax = b$ is defined as $\kappa(A) = \|A\| \|A^{-1}\|$ with respect to some matrix-norm $\|\cdot\|$. It can take values between 1 and infinity and in very rough terms states how much can a perturbation ΔA in the matrix A or perturbation Δb in the b -side of the system affect Δx , the change in the solution. The lower the condition number, the better conditioned the system of linear equations will be. Infinite condition number implies that the matrix A is singular. For more precise definition see for instance [7].

Minimizing the condition number of the matrix S is advantageous for several reasons. Matrix T which contains all measured or predicted spectra is only an approximation of the real spectra of the different peptides. A good-conditioned sub-matrix S would lie far from singularity. This would imply that the correct but practically unobtainable matrix $S' = S + \Delta S$ would give a result $x + \Delta x$, not far from our result x . The same holds for the measured ion counts which form the observation vector c . Our measurements are very inexact, therefore, a system of linear equations that is robust with regard to perturbations is beneficial. A low condition number also implies sufficient orthogonality between different vectors, which is also one of the natural goals.

We can now define the computational problem

Problem 1 (Best conditioned submatrix). *Given a $n \times m$ matrix T , select an $m \times m$ submatrix S^* such that S^* has the best condition number out of all $m \times m$ submatrices of T .*

In the next section, we show that the problem of finding such a best conditioned submatrix is NP-hard. Finding best submatrices for other objectives is sometimes hard, sometimes easy, depending on the objective. For example, for any nontrivial hereditary property that also holds for permutation matrices, the problem is hard [8], while finding a submatrix of a given $\|\cdot\|_1$ or $\|\cdot\|_\infty$ norm is easy.

2 Best Conditioned Submatrix Is NP-Hard

We will now prove that the problem of selecting k rows from a $n \times m$ matrix A in such a way that the condition number of the resulting $k \times m$ matrix is minimal (a generalization of the best conditioned submatrix problem), is NP-hard in its decision form. We will do this via a reduction from the maximum clique problem. First, we establish some simple facts regarding the condition number of a matrix.

The condition number of a rectangular matrix A is defined as

$$\kappa(A) = \frac{\max_{\|x\|=1} \|Ax\|}{\min_{\|x\|=1} \|Ax\|} \quad (2)$$

where x is a vector of unit norm. It can be thought of as the ratio of maximum to minimum “magnification” of the transformation A . The following lemmas can be proved by simple algebraic manipulation and a selection of convenient vectors to represent bounds.

Lemma 1. *Let A be a $n \times k$ matrix with $k \leq n$ and all column vectors of norm between 1 and $(1 - \varepsilon)^{-1}$. Then the condition number of the matrix A is at least $\kappa(A) \geq (1 - \varepsilon)\sqrt{1 + \cos \alpha}$, where α is the angle between any of its column vectors a_i and a_j .*

Lemma 2. *Let A be an orthogonal $n \times k$ matrix, $k \leq n$. Let a_i and a_j be the column vectors of A with maximal resp. minimal norm. Then $\kappa(A) = \|a_i\| / \|a_j\|$.*

Corollary 1. *Let A be a $n \times k$ matrix, $k \leq n$. Furthermore, let all k column vectors of A have unit length. Then, the condition number of A is equal to 1 if and only if all k column vectors of A are pairwise orthogonal.*

Proof. Follows from Lemmas 1 and 2. □

The last step before proving the reduction is the following theorem. The proof will be constructive and will give an algorithm with polynomial running time to construct the described assignment.

Theorem 1. *Given a graph G of n vertices, we can assign each vertex in G a vector of dimension n in such a way that the vectors of vertices neighboring in the graph are orthogonal and the vectors of non-neighboring vertices are not. Furthermore, we can do this in time polynomial in the number of vertices so that the following conditions hold:*

1. *all vectors have rational coefficients, that can be represented in size polynomially bounded in n ;*

2. for any given ε , the norms of all vectors are between 1 and $(1 - \varepsilon)^{-1}$;
3. for some $\beta > 0$ that depends only on n , the angle between any two vectors is larger than β ;
4. for some $\alpha > \beta$ that depends only on n , any two vectors will be either orthogonal, or their angle is smaller than α .

Proof. In order to simplify the proof we will enforce linear independence between all vectors. We will proceed by induction on the size n of the graph. If $n = 1$, the claim is trivial. Suppose that the claim is correct for all graphs of size $n - 1$. Hence, given a graph G of n vertices, we can select $n - 1$ vertices and construct the corresponding $(n - 1)$ -dimensional vectors $x'_1, \dots, x'_{n-1}$ that fulfill the conditions 1–4 and linear independence. We now show how to define the n -dimensional vectors $x_1, \dots, x_{n-1}$ and how to construct a n -dimensional vector z that satisfies the statement.

For $i = 1, \dots, n - 1$, we set $(x_i)_j := (x'_i)_j$, $1 \leq j \leq n - 1$, and $(x_i)_n = 0$, that is, we take the vector x'_i , we increase its dimension by one, and we set the last component to zero. Notice that the new vectors $x_1, \dots, x_{n-1}$ are still linearly independent and the angle between any two vectors x_i and x_j is the same as the angle between the corresponding vectors x'_i and x'_j , $1 \leq i < j \leq n - 1$.

At this point, we can define the vector z that corresponds to the vertex v_n of G . To do so, we need to enforce three sets of constraints. Let us suppose without loss of generality that the vectors corresponding to the vertices adjacent to v_n in G are labeled x_1 to x_i , and that the vectors non-adjacent to v_n are labeled x_{i+1} to x_{n-1} . We need to set the scalar product of z and x_k , $1 \leq k \leq i$, to be zero and the scalar product of z and x_k , for $i + 1 \leq k \leq n - 1$, to be non-zero, in order for the orthogonality resp. non-orthogonality constraints to be satisfied. The last constraint is the linear independence constraint. If the set of vectors x_1 to x_{n-1} and z is linearly independent, the determinant of the matrix they form will be non-zero. We expand the determinant as a linear combination of minors with respect to the vector z . All of these constraints now form a set of linear equations

$$\begin{array}{ccccccc} x_{11}z_1 & + & \dots & + & x_{1n}z_n & = & b_1 \\ x_{21}z_1 & + & \dots & + & x_{2n}z_n & = & b_2 \\ \vdots & & & & \vdots & & \vdots \\ x_{(n-1)1}z_1 & + & \dots & + & x_{(n-1)n}z_n & = & b_{n-1} \end{array} \quad (3)$$

$$\det \begin{pmatrix} x_{12} & \dots & x_{1n} \\ \vdots & & \vdots \\ x_{(n-1)2} & \dots & x_{(n-1)n} \end{pmatrix} z_1 - \dots + (-1)^{n+1} \det \begin{pmatrix} \dots \end{pmatrix} z_n = b_n$$

If we show that the matrix that corresponds to the left-hand side is invertible, we find a solution for z for every choice of the vector $b = (b_1, \dots, b_n)$. This is simple to see if we do another Laplacian expansion of the determinant, again according to the last line. We get

$$\det(A) = \sum_{l=1}^n ((-1)^{l+1} \det A_l)^2 > 0 \quad (4)$$

where A_l , $1 \leq l \leq n$ are the corresponding minors. The vectors $x_1, \dots, x_{n-1}$ are linearly independent, hence at least one of the minors has non-zero determinant. We need to choose the vector b so that the conditions 3 and 2 are fulfilled.

The determinant of a matrix is the volume of the parallelepiped with sides given by the vectors of the matrix. When we modify the existing vectors and we add a new vector as we did, we increase the dimension by one and the new determinant will be the old determinant multiplied by the norm of the vector z and $\sin \phi$, where ϕ is the angle between the vector space spanned by $x_1, \dots, x_{n-1}, z$. Define the angle ϕ_i to be the angle between the vectors x_i and z , $1 \leq i \leq n-1$. If we consider only the vectors b , for which $b_i \geq 0$, $1 \leq i \leq n$ holds, we get $\phi \leq \phi_i$, $1 \leq i \leq n-1$, as $\phi_i \leq \pi/2$.

Due to the aforementioned interpretation of the determinant, if we set $b_n = \det(x'_1, \dots, x'_{n-1})$, we get $\|z\| = 1/\sin \phi$. Recall that we need to set $b_k = 0$, $1 \leq k \leq i$. On the other hand $\langle z, x_k \rangle = \|x_k\| \cdot \|z\| \cos \phi_k$, $i+1 \leq k \leq n-1$. We will now set all the non-zero b -components to $\cot \beta \cdot \|x_k\|$, for a certain β chosen later. This will force all ϕ_k to be equal, and larger than β . In fact, for $i+1 \leq k \leq n-1$: $\|z\| \cdot \|x_k\| \cos \phi_k = \cot \beta \cdot \|x_k\|$ implies $\cos \phi_k = \cot \beta \sin \phi$ by using $\|z\| = 1/\sin \phi$. Further, since $\phi \leq \phi_i$ and the $\cot$ function is decreasing in the interval $(0, \pi/2)$, we immediately get $\beta \leq \phi_k$. The claim is trivial for the angle corresponding to orthogonal vectors. Thus Condition 3. is proved.

We need to prove that $\phi_k \leq \alpha$, for $i+1 \leq k \leq n-1$. Suppose that β and α are chosen sufficiently large, so that

$$\sin \phi \geq \cos \alpha \cdot \tan \beta \quad (5)$$

holds. Then, by taking advantage of $\|z\| \cos \phi_k = \cot \beta$ it is an easy algebraic manipulation to conclude that $\phi_k \leq \alpha$, settling the claim.

Let us now try to give a bound for the angle ϕ , so that (5) can be fulfilled by some angles β and α . We know that ϕ is the angle between z and orthogonal projection of z onto the space spanned by $x_1, \dots, x_{n-1}$. This projection is a linear combination $\sum_i \lambda_i x_i$ of vectors x_i , with a selection of λ_i that minimizes the angle ϕ . We can also select λ so that $\|\sum_i \lambda_i x_i\| = 1$. Then, by the linearity of the scalar product:

$$\begin{aligned} \cos \phi &= \frac{1}{\|\sum_i \lambda_i x_i\|} \frac{1}{\|z\|} \langle \sum_i \lambda_i x_i, z \rangle = \frac{1}{\|z\|} \sum_i \lambda_i \langle x_i, z \rangle = \\ &= \frac{1}{\|z\|} \sum_i \lambda_i \|x_i\| \|z\| \cos \phi_i \leq 2 \cos \phi_j \sum_i \lambda_i \end{aligned} \quad (6)$$

The last inequality follows from the fact that all ϕ_i are equal and $\|x_i\| \leq 1 + \varepsilon \leq 2$, for ε small. Hence, we need to bound $\sum_i \lambda_i$. First, some remarks are in order. The expression $\sum_i \lambda_i x_i$ gives us a vector $\lambda = (\lambda_1, \dots, \lambda_{n-1}, 0)$ according to the basis $x_1, \dots, x_{n-1}, z$ (recall that these vectors are linearly independent). The matrix $X := (x_1, \dots, x_{n-1}, z)$ describes the transformation of a vector from the basis $x_1, \dots, x_n$ to the canonical basis. The norm of the vector $\sum_i \lambda_i x_i$ is

one, so $\|X\lambda\| = 1$. Since the norm of a matrix X is defined as $\max_{\|x\|=1} \|Xx\|$ and $\|x\|_1 = \sum_i |x_i| \leq \sqrt{n} \|x\|$, we get:

$$\sum_{i=1}^n \lambda_i \leq \|\lambda\|_1 \leq \sqrt{n} \|\lambda\| \leq \sqrt{n} \|X^{-1}\lambda'\| \leq \sqrt{n} \frac{1}{\|X\|} \leq \frac{\sqrt{n}}{(1-\varepsilon)\sqrt{1+\cos\alpha}} \leq \sqrt{n} \quad (7)$$

The fact that $\|X\| \geq (1-\varepsilon)\sqrt{1+\cos\alpha}$ follows from the proof of Lemma 1 and α will be selected such that $\sqrt{1+\cos\alpha} > (1-\varepsilon)^{-1}$.

From (6), (7) and the fact that $\beta \leq \phi_i$ we have:

$$\cos\phi < 2\sqrt{n}\cos\beta \quad (8)$$

If we choose β to be large enough, such that $2\sqrt{n}\cos\beta = 1/2$ holds, we force $\phi \geq \pi/3$. Additionally, $\cot\beta$ needs to be a rational number with a small representation, but this is easy because the cot-function is surjective and continuous. By choosing α sufficiently large, (5) is also satisfied and Condition 4. is true. Estimates for both β and α are clearly computable in polynomial time and with the only knowledge of n , due to the asymptotic behavior of the cos- and tan-functions.

To fulfill Conditions 1. and 2., notice that the coefficients of z are rational and polynomially bounded, because it is a solution of a system of linear equations with rational coefficients. Further, we need to normalize the vector z , whose norm might otherwise be irrational. For the vectors z , we need to determine the norm $\|z\| = \sqrt{\sum_{i=0}^n z_i^2}$ with some precision ε . There exist a number of well-known algorithms which can for a given s and ε find a number y , such that $\sqrt{s}(1-\varepsilon) \leq y \leq \sqrt{s}$. Most can do this in time at worst linear in the number of digits of s and the precision of ε . For the lengths of the normalized vectors, it will then hold:

$$\begin{aligned} \left\| \frac{z}{\|z\|} \right\| &\leq \left\| \frac{z}{y} \right\| \leq \left\| \frac{z}{\|z\|(1-\varepsilon)} \right\| \\ 1 &\leq \left\| \frac{z}{y} \right\| \leq \frac{1}{1-\varepsilon} \end{aligned} \quad (9)$$

This settles Conditions 1. and 2. and thus finishes the proof. Since we deal only with rational numbers with a small representation and we need to solve n systems of linear equations, the algorithm runs in polynomial time. $\square$

The problem of representing a graph by one vector for each node and letting orthogonality between vectors determine edges has been studied previously [9]. The proposed construction comes close, but can, unfortunately, not be used directly for our purposes, since we need conditions 2 to 4 in Theorem 1 to guarantee that our construction will run in polynomial time.

We are now ready to prove the actual reduction.

Theorem 2. *Given a $n \times m$ matrix A and a number k , $k \leq m$, and a bound $\chi(n)$, the problem of finding a $n \times k$ submatrix of A such that it has condition number smaller than $\chi(n)$ is NP-hard.*

Proof. We prove the claim by a reduction from the maximum clique problem, which is known to be NP-complete [10].

Let graph G be an instance of the maximum clique problem on n vertices. To each vertex in G we assign a vector of norm between 1 and $(1 - \varepsilon)^{-1}$ from the Euclidean space $\mathbb{R}^n$. Furthermore the vectors corresponding to vertices adjacent in the graph G are orthogonal, and vectors corresponding to non-adjacent vertices are not orthogonal. Such an assignment can be found in polynomial time by Theorem 1, Conditions 2 – 4 will rule out almost-orthogonal vectors.

Now, we will be able to find a $n \times k$ submatrix of low enough condition number, if and only if G has a clique of size k . We need to select ε in such a way that we will be able to distinguish between matrices that are orthogonal and matrices that are not, based on their condition number. By Lemma 2, if a matrix is orthogonal, its condition number will be the ratio of its largest to its smallest vector. In the worst case, this can be $(1 - \varepsilon)^{-1}$. If the matrix is not orthogonal, the angle between at least one pair of its vectors is smaller than α and therefore its condition number will be at least $(1 - \varepsilon)\sqrt{1 + \cos \alpha}$ by Lemma 1. Notice that α depends on n and is given by Theorem 1. We need to set ε so that $(1 - \varepsilon)^{-1} < (1 - \varepsilon)\sqrt{1 + \cos \alpha}$ holds (this can always be achieved for small enough ε). $\chi(n)$ is then a threshold function that fulfills :

$$(1 - \varepsilon)^{-1} < \chi(n) < (1 - \varepsilon)\sqrt{1 + \cos \alpha}.$$

If the best conditioned submatrix consists of a set of columns with condition number smaller than $\chi(n)$, we can be certain that this matrix corresponds to a clique in the graph G according to the Corollary 1. If the condition number of the selected submatrix is larger, we can be certain that there is no clique of size k in the graph. $\square$

3 A Heuristic Solution of the Transition Selection Problem

Since the best conditioned submatrix problem is NP-hard, we cannot expect being able to solve it exactly. Therefore we propose a heuristic with the goal of finding submatrices with very good condition numbers. In the context of many problems, such as solving systems of linear equations, this is sufficient.

The problem of numerical stability occurs very often in matrix decompositions. It is usually countered by pivoting, i.e. reordering of rows (or columns) of the matrix. Pivoting is usually found by a greedy algorithm, which effectively reduces the condition number of the most significant parts of the solution. We propose to exploit such a pivoting procedure to select those rows (columns) which reduce the condition number of the resulting system of linear equations. A number of different pivoting procedures could be used.

As an example, we can demonstrate the finding of a suitable pivoting using the pivoted Gram-Schmidt process for matrix QR -decomposition. Such a decomposition gives us a permutation matrix P , orthogonal matrix Q and upper

triangular matrix R such that $AP = QR$. The matrices Q and R can be disregarded as the matrix P is the principal output. The matrix A is decomposed with n column vectors of length m . The goal is to select m of these vectors a_{i_1} to a_{i_m} to form a $m \times m$ matrix. We will use a modification of the Gram-Schmidt orthogonalization process on row vectors of the matrix A .

First let us define the projection of one vector upon another as $proj_{ab} = \langle a, b \rangle b \|b\|^{-2}$. Now the modified algorithm will find the permutation matrix P in m steps. The matrix A will be modified while generating the matrix P . First, the norm of every row vector a_i is computed. The row vectors are denoted $norm_i = \|a_i\|$. In the i -th step we now choose the vector with the maximum norm from vectors $a_i \dots a_n$ and swap it with the vector a_i . This vector would be our next base vector if we were constructing an orthogonal base in the original algorithm. Now we subtract the vector component contributed by t_i from the remaining vectors a_{i+1} to a_n and update their norms. After m steps, we have chosen m vectors and these will form the matrix A .

The time complexity of the algorithm is $O(nm^2)$. This is easily seen as we are choosing the vectors in m steps. In every step, we need to subtract projections from each of the at most n vectors. Subtracting projections takes $O(m)$ time.

We can use a number of partial or even complete pivoting schemes; the Gram-Schmidt QR decomposition was chosen as it is well known and easy to understand. This approach is also very flexible: We might for instance scale each $norm_i$ by a function f_i , which reflects our willingness to choose this row with regard to other constraints of the particular problem being solved.

4 Quantification Workflow and Experimental Results

Our method of quantifying peptides will differ from the currently standard method. The first problem that needs to be tackled is the availability of spectra of all peptides. Spectra that are available from even the biggest collections cover just a small part of all the possible peptides for the corresponding organisms. Particularly spectra of peptides that are usually low abundant in mixtures are rare. In order to get our peptide spectra database (matrix T), we first design and train a simple hidden Markov model similar to [11]. In our case the spectrum model is used to predict spectra, but not to identify peptides from spectra. We augmented the existing database by predicted spectra of all peptides. Thus, we obtained a spectrum database for all possible peptides of the given organism. Detailed explanation of the process is out of the scope of this paper, since spectrum prediction is a topic of its own.

In our experiments we generated random hypothetical biological samples. We sampled ion count for each peptide and picked randomly a small number of peptides that are targeted for quantification. Since we know the actual composition of the sample, we can compare the estimate by both the standard method and our method with the true ion counts. One of our side goals is to measure as few transitions as possible. One extreme is measuring only as many transitions as there are peptides to be quantified: this is the currently standard method. Another extreme is measuring all the possible transitions and solving a least squares

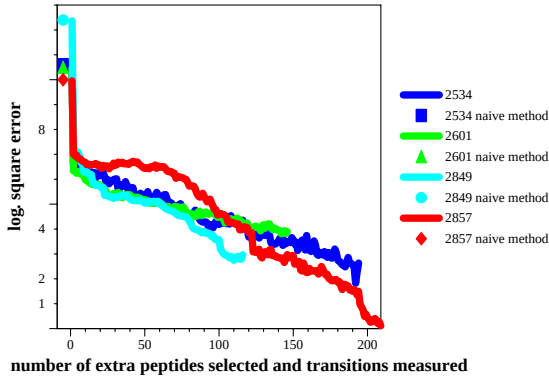


Fig. 1. Error decrease for peptides of 4 different masses (denoted in legend). Note that the maximum number of measured peptides is variable due to different peptide numbers for different masses. Errors for the standard method are depicted by marks.

problem. This method is infeasible due to time and hardware constraints. In the experiments, we chose different numbers of measured transitions: one per peptide that needs to be quantified up to the maximum number of different transitions for the appropriate peptide mass. In addition to the method described in the section before, we will estimate the ion count of more than just the desired peptides. If the number of the transitions measured per peptide is larger than one, we are able to quantify additional, previously unselected, peptides. For each number of transitions we first chose the additional peptides to be quantified. We did this by a simple heuristic which always chose the peptide with the most conflicting spectrum with regard to our already selected peptides. Then we performed a SVD decomposition of the matrix formed by all spectra of the selected peptides. The SVD decomposition provides us with a pivoting analogous to the Gram-Schmidt pivoting from the previous section. We then chose those transitions which are permuted to the front by the pivoting.

The results of our “in silico” experiments look promising. It seems that the additional quantification of only a few peptides already increases the quantification accuracy dramatically. The graph of average square error of quantification of 10 randomly selected peptides from 50 different mixtures for 4 peptides with 4 different peptide masses is depicted on a logarithmic scale in Figure 1. The average error of the standard method is depicted by marks near the y axis.

5 Discussion and Open Questions

Summary. In this paper, we introduced the problem of optimal transition selection in the context of targeted proteomics. We proposed to model the problem by selecting and solving a system of linear equations with low condition number. We show the problem of finding such a system of linear equations with low condition number to be NP-hard.

MRM and targeted proteomics. In order to increase accuracy and allow for the quantification of low abundant peptides, we propose a framework which is based on a suitable pivoting procedure. The solution is very flexible and allows us to incorporate additional information on the biotechnological system. A lot of practical points are still open and have to be solved when going towards a quantification of the whole proteome.

We performed a set of experiments to simulate the quantification accuracy of our method. The results are very encouraging, however experiments with real data need to be designed and performed in order to validate the method.

Acknowledgments. The authors would like to thank Christian Ahrens, Sacha Baginsky, Katja Bärenfaller, Erich Brunner, and Vinzenz Lange for discussions concerning biological nature of the paper. For direct and indirect discussions on algebraic topics the authors are grateful to Peter Arbenz, Zlatko Drmač, and Miroslav Fiedler. We would also like to thank the anonymous referees for pointing out previous work [9] related to our problem.

References

1. Gygi, S., Rist, B., Gerber, S., Turecek, F., Gelb, M., Aebersold, R.: Quantitative analysis of complex protein mixtures using isotope-coded affinity tags. *Nature Biotechnology* 17, 994–999 (1999)
2. Ross, P., Huang, Y., Marchese, J., et al.: Multiplexed Protein Quantitation in *Saccharomyces cerevisiae* Using Amine-reactive Isobaric Tagging Reagents. *Molecular & Cellular Proteomics* 3(12), 1154–1169 (2004)
3. Listgarten, J., Emili, A.: Statistical and Computational Methods for Comparative Proteomic Profiling Using Liquid Chromatography-Tandem Mass Spectrometry. *Molecular & Cellular Proteomics* 4(4), 419–434 (2005)
4. Fischer, B., Grossmann, J., Roth, V., Gruissem, W., Baginsky, S., Buhmann, J.: Semi-supervised LC/MS alignment for differential proteomics. *Bioinformatics* 22(14) (2006)
5. Fischer, B., Roth, V., Buhmann, J.: Time-series alignment by non-negative multiple generalized canonical correlation analysis, feedback (2008)
6. Šrámek, R., Fischer, B., Vicari, E., Widmayer, P.: Optimal Transitions for Targeted Protein Quantification: Best Conditioned Submatrix Selection, <http://www.pw.inf.ethz.ch>
7. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis, 3rd edn. Springer, Heidelberg (2002)
8. Bartholdi, J.: A Good Submatrix is Hard to Find. *Operations Research Letters* 1(5), 190–193 (1982)
9. Lovász, L., Saks, M., Schrijver, A.: Orthogonal Representations and Connectivity of Graphs. *Linear Algebra Applications* 114–115, 439–454 (1989)
10. Karp, R.: Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, 85–103 (1972)
11. Fischer, B., Roth, V., Roos, F., Grossmann, J., Baginsky, S., Widmayer, P., Gruissem, W., Buhmann, J.: NovoHMM: A Hidden Markov Model for de Novo Peptide Sequencing. *Analytical Chemistry* 77(22), 7265–7273 (2005)

Computing Bond Types in Molecule Graphs

Sebastian Böcker^{1,2}, Quang B.A. Bui¹, Patrick Seeber¹, and Anke Truss¹

¹ Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany

sebastian.boecker@uni-jena.de

{bui,patrick.seeber,truss}@minet.uni-jena.de

² Jena Centre for Bioinformatics, Jena, Germany

Abstract. In this paper, we deal with restoring missing information in molecule databases: Some data formats only store the atoms' configuration but omit bond multiplicities. As this information is essential for various applications in chemistry, we consider the problem of recovering bond type information using a scoring function for the possible valences of each atom—the BOND TYPE ASSIGNMENT problem. We prove the NP-hardness of BOND TYPE ASSIGNMENT and give an exact fixed-parameter algorithm for the problem where bond types are computed via dynamic programming on a tree decomposition of the molecule graph. We evaluate our algorithm on a set of real molecule graphs and find that it works fast and accurately.

1 Introduction

The *structural formula* of a chemical compound is a representation of the molecular structure, showing both how the atoms are arranged and the chemical bonds within the molecule. Throughout this paper, we refer to structural formula as *molecule graph*. An important information on a molecule graph is the bond type. It is essential for many applications in chemistry, for example to compute the molecular mechanics force field [12]. Unfortunately, this information is omitted by many data formats that represent molecule graphs, such as Gaussian file formats and Mopac file formats, and even by the widely used Protein Data Bank format *pdb* [12]. Moreover, in combinatorial chemistry, the backbone of the molecule (skeletal formula) may be drawn either manually or automatically, again omitting bond types. Now the question is how to reassign such bond types to the molecule graph. This problem is aggravated by the fact that several elements can have different states of valence, enabling them to form different numbers of bonds.

Previous work. Several heuristic approaches have been introduced for this problem [8, 12, 3]. An integer linear programming based algorithm can solve the problem exactly for small molecules [6]. Recently, Dehof *et al.* introduced a running time heuristic for this problem (paper in preparation).

Our contribution. In this paper, we prove that the problem of reassigning bond types to molecule graph is NP-hard. We then introduce a *tree decomposition-based* algorithm that computes an exact solution for the problem with running

time $O(a_{max}^{2\omega} \cdot 3^\alpha \cdot \omega \cdot m + m)$, where m is the number of nodes in the tree decomposition of the molecule graph, a_{max} is the maximum *open valence* of an atom, d is the maximum degree of an atom in the molecule, $\omega - 1$ is the treewidth of the molecule graph, and $\alpha := \min\{\binom{\omega}{2}, \omega d\}$. With our algorithm, we prove that the problem of reassigning bond types to molecule graphs is *fixed-parameter tractable* (FPT) [5, 10] in the treewidth of the molecule graph and the maximum open valence of an atom in the molecule. Furthermore, we implemented our algorithm and evaluated it on real molecules of the MMFF94 dataset. As we expected, the treewidth of molecule graph is rather small for biomolecules: for all molecule graphs in our dataset, the treewidth is at most three, and our algorithm solves the problem mostly in well under a second.

2 Definitions

A *molecule graph* is a graph $G = (V, E)$ where each vertex in V corresponds to an atom and each edge in E corresponds to a chemical bond between the respective atoms. We denote an edge from u to v by uv . For every vertex $v \in V$, let $\mathcal{A}_v \subseteq \{1, \dots, max\_valence\}$ be the set of *valences* of the atom at vertex v , where *max_valence* denotes the maximum valence of an atom in the molecule. Then $A_v := \mathcal{A}_v - \deg(v) = \{a - \deg(v) : a \in \mathcal{A}_v, a - \deg(v) \geq 0\}$ is the set of admissible *open valences* we can still assign to v . We set $A_v^* := \max A_v$.

Let $b : E \rightarrow \{0, 1, 2\}$ be a weight function assigning a bond type, represented by the bond multiplicity lowered by one, to every bond $uv \in E$. We call such b an *assignment*. An assignment b *determines* a valence $x_b(v) := \sum_{u \in N(v)} b(uv) + \deg v$ for every vertex $v \in V$, where $N(v)$ denotes the set of neighbors of v . The assignment b is *feasible* if $x_b(v) \in \mathcal{A}_v$ for every vertex $v \in V$.

A *scoring function* $S_v : \mathcal{A}_v \rightarrow \mathbb{R}^+$ assigns a finite positive score $S_v(a)$ to a valence $a \in \mathcal{A}_v$ at vertex v . The score $S(b)$ of an assignment b is

$$S(b) := \sum_{v \in V} S_v(x_b(v)). \quad (1)$$

Given a molecule graph $G = (V, E)$, our task is to find a feasible assignment b for G with minimum score $S(b)$.

For open valences, we define a scoring function $s_v : \{0, \dots, A_v^*\} \rightarrow \mathbb{R}^+$ via $s_v(a) := S_v(a + \deg v)$ for $a + \deg v \in \mathcal{A}_v$, and $s_v(a) := \infty$ otherwise. We can express (1) in terms of open valences: For an assignment b , the atom at vertex v take valence $y_b(v) := \sum_{u \in N(v)} b(uv)$ and we can compute

$$S(b) = \sum_{v \in V} s_v(y_b(v)).$$

By this definition, $S(b) = \infty$ holds for assignments b that are not feasible.

In this paper, we mostly work with the abovementioned open valences. Therefore open valence is referred to as *valence* for simplicity. The BOND TYPE ASSIGNMENT PROBLEM is defined as follows:

Bond Type Assignment Problem. Given a molecule graph $G = (V, E)$ with (open) valence sets $A_v \subseteq \{1, \dots, a_{max}\}$ and scoring functions s_v for every $v \in V$. Find a feasible assignment b for G with minimum score $S(b)$.

In Sect. 3, we show that the BOND TYPE ASSIGNMENT problem is NP-hard even if every vertex of the molecule graph is of degree at most three and the atom at every vertex has a valence of at most four.

3 Hardness of the Problem

Given an input for the BOND TYPE ASSIGNMENT problem, the BOND TYPE ASSIGNMENT *decision* problem asks if there is a feasible assignment b for the input graph. In this section, we show that this problem is NP-hard. Therefore, BOND TYPE ASSIGNMENT is also NP-hard.

Theorem 1. *The BOND TYPE ASSIGNMENT decision problem is NP-hard, even on input graphs where every vertex has degree at most three and atom valences are at most four.*

In our hardness proof, we will use reduction from a variant of the 3-SAT problem:

Definition (3-SAT). Given a set X of n boolean variables $\{x_1, \dots, x_n\}$ and a set C of m clauses $\{c_1, \dots, c_m\}$. Each clause is a disjunction of at most three literals over X , for example $(x_1 \vee \overline{x_2} \vee x_3)$. Is there an assignment $X \rightarrow \{true, false\}$ that satisfies all clauses in C , i.e., at least one literal in every clause is *true*?

Definition (3-SAT*). The variant of 3-SAT where every variable occurs at most three times and every literal at most twice is called the 3-SAT* problem.

To transform a 3-SAT problem instance into a 3-SAT* problem instance, we first replace every t -th occurrence, $t \geq 3$, of a variable with a new variable and some auxiliary clauses of length two, which assure that new variables are consistently assigned. Then we remove all variables that only occur in either positive or negative literals and clauses containing such variables. The resulting 3-SAT instance is a 3-SAT* instance and it is satisfiable if and only if the original 3-SAT instance is satisfiable. This 3-SAT* instance is only larger than the original 3-SAT instance by a polynomial factor. Since 3-SAT is NP-hard [7], the 3-SAT* problem is also NP-hard.

Proof (of Theorem 1). By a polynomial reduction from 3-SAT* to the BOND TYPE ASSIGNMENT decision problem, we will show that the BOND TYPE ASSIGNMENT decision problem is NP-hard, even if every vertex is of degree at most three and valence at most four.

Given a 3-SAT* formula, we can safely discard all clauses containing variables that only occur in either positive or negative literals. Afterwards, every variable occurs at least twice and at most three times in at least one positive and one negative literal. We then construct the SAT-graph $G = (V, E)$ for the BOND TYPE ASSIGNMENT problem as follows:

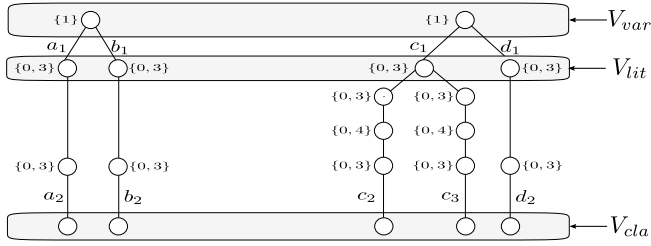


Fig. 1. The building blocks of G . The unmasked nodes are auxiliary vertices. The valence sets of the clause vertices are not shown here.

The vertex set V consists of four subsets V_{var} , V_{lit} , V_{cla} and V_{aux} . For each variable x_i of the 3-SAT* instance, the vertex set V_{var} contains a *variable vertex* v_i and the vertex set V_{lit} contains two *literal vertices* u_i and u'_i corresponding to the literals x_i and $\overline{x_i}$. The set V_{cla} contains, for every clause c_j of the 3-SAT* instance, a *clause vertex* w_j . Finally, we need a couple of auxiliary vertices subsumed in V_{aux} as shown in Fig. 1.

The valence set of each variable vertex is $\{1\}$, of each literal vertex $\{0, 3\}$, and of a clause vertex $\{1, \dots, d\}$, where $d \leq 3$ is the number of literals contained in the corresponding clause. The valence sets of auxiliary vertices are set as shown in Fig. 1. We use the trees shown in Fig. 1 as building blocks to connect the vertices of G .

If both literals of a variable occur once, we connect each of the literal vertices to the clause vertex that corresponds to the clause containing this literal via an auxiliary vertex with valence set $\{0, 3\}$. See Fig. 1(left).

If one literal of a variable occurs once and the other twice, we connect the literal vertex, which corresponds to the literal occurring in one clause, to the corresponding clause vertex via an auxiliary vertex with valence set $\{0, 3\}$. The literal vertex corresponding to the literal occurring in two clauses is connected to each of the corresponding clause vertices via a chain of three auxiliary vertices with valence sets $\{0, 3\}$, $\{0, 4\}$, $\{0, 3\}$. See Fig. 1 (right).

Before proving that the constructed BOND TYPE ASSIGNMENT instance has a feasible assignment if and only if 3-SAT* instance is satisfiable, we consider the two building blocks of G shown in Fig. 1. Let $a_1, a_2, b_1, b_2, c_1, c_2, c_3, d_1, d_2$ denote the bond type of the corresponding edges as shown in Fig. 1. In a feasible assignment of G , the following facts can be easily observed:

The bond types $a_1, a_2, b_1, b_2, c_1, c_2, c_3, d_1, d_2$ can take a value of one or two. The bond type two can only be assigned to either a_1 or b_1 , and to either c_1 or d_1 , and the corresponding literal vertex has to take valence three, the other one has to take valence zero. Furthermore, it holds that $a_1 = a_2$, $b_1 = b_2$, $c_1 = c_2 = c_3$ and $d_1 = d_2$.

The fact that exactly one of two edges incident to a variable vertex is a double binding models the rule that exactly one of the literals $x_i, \overline{x_i}$ of a variable x_i is satisfied. The valence of a clause vertex takes a value of at least one if and only if the corresponding clause contains literals whose literal vertices have valence

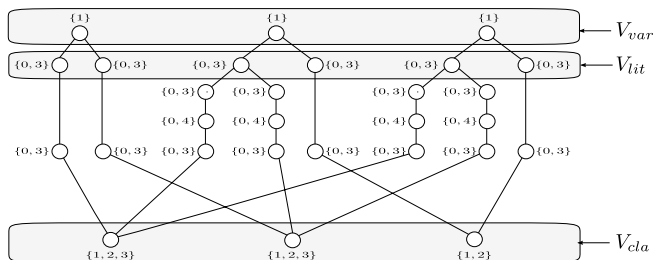


Fig. 2. The unmasked vertices are auxiliary vertices. The variable vertices represent variables x_1, x_2, x_3 from left to right. The literal vertices represent literals $x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3$ from left to right. The clause vertices represent clauses $(x_1 \vee x_2 \vee x_3)$, $(\bar{x}_1 \vee x_2 \vee x_3)$, $(\bar{x}_2 \vee \bar{x}_3)$ from left to right.

three. This implies that a clause is satisfied if and only if it contains a *true* literal. Furthermore, the valence set $\{1, \dots, d(w)\}$ of a clause vertex w forces any algorithm for the BOND TYPE ASSIGNMENT problem to assign a double binding to at least one of the edges incident to w . This implies that at least one of the literals contained in each clause has to be *true*.

Therefore, there is a feasible solution for the constructed BOND TYPE ASSIGNMENT instance if and only if the 3-SAT* instance is satisfiable. Since the reduction can be done in polynomial time and the 3-SAT* problem is NP-hard, the BOND TYPE ASSIGNMENT decision problem is also NP-hard. $\square$

4 The Algorithm

To solve the BOND TYPE ASSIGNMENT problem exactly, we use the dynamic programming approach on the *tree decomposition* of the input graph [11]. In the following subsection, we give a short introduction to the tree decomposition concept. We follow Niedermeier’s monograph [10] in our presentation.

4.1 Tree Decompositions

Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i \mid i \in I\}, T \rangle$ where each X_i is a subset of V , called a bag, and T is a tree containing the elements of I as nodes and the three following properties must hold:

1. $\bigcup_{i \in I} X_i = V$;
2. for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$; and
3. for all $i, j, k \in I$, if j lies on the path between i and k in T then $X_i \cap X_k \subseteq X_j$.

The *width* of the tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ equals $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of G is the minimum number $\omega - 1$ such that G has a tree decomposition of width $\omega - 1$.

Given a molecule graph G , we first compute the tree decomposition T of G before executing our algorithm on T to solve the BOND TYPE ASSIGNMENT

problem on G . As we show later, the running time and the required space of our algorithm grow exponentially with the treewidth of G . Therefore, the smaller the width of the tree decomposition of G , the better running time our algorithm will achieve. Unfortunately, computing a tree decomposition with minimum width is an NP-hard problem [2]. But our input graphs usually show a particular structure that allows us to efficiently compute optimal tree decompositions.

A graph is called *outerplanar* if it admits a crossing-free embedding in the plane such that all vertices are on the same face. A graph is *1-outerplanar* if it is outerplanar; and it is *r -outerplanar* for $r > 1$ if, after removing all vertices on the boundary face, the remaining graph is an $(r - 1)$ -outerplanar graph. Every r -outerplanar graph has treewidth at most $3r - 1$ [4], and we can compute optimal tree decompositions of r -outerplanar graphs in time $O(r \cdot n)$ [1], where n is the number of vertices in the graph. The important observation here is that most molecule graphs of biomolecules are r -outerplanar for some small integer r , such as $r = 2$. For such molecules, we can first compute the optimal tree decomposition of the molecule graph, and since the treewidth is rather small, our tree decomposition-based algorithm can be used to solve the BOND TYPE ASSIGNMENT problem efficiently.

To improve the legibility and to simplify description and analysis of our algorithm, we use *nice tree decompositions* instead of arbitrary tree decompositions in the remaining part of this paper. Here, we assume the tree T to be rooted. A tree decomposition is a *nice tree decomposition* if it satisfies the following conditions:

1. Every node of the tree has at most two children.
2. If a node i has two children j and k , then $X_i = X_j = X_k$; in this case i is called a *join node*.
3. If a node has one child j , the one of the following situations must hold:
 - (a) $|X_i| = |X_j| + 1$ and $X_j \subset X_i$; in this case X_i is called an *introduce node*.
 - (b) $|X_i| = |X_j| - 1$ and $X_i \subset X_j$; in this case X_i is called a *forget node*.

After computing a tree decomposition of width k and m nodes for the input graph G , we transform this tree decomposition into a nice tree decomposition with the same treewidth and $O(m)$ nodes in linear time using the algorithm introduced in [9] (Lemma 13.1.3). Then we execute our algorithm on the nice tree decomposition to compute the optimal bond type assignment for G .

4.2 Tree Decomposition-Based Algorithm

Assume that a nice tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ of width $\omega - 1$ and $O(m)$ nodes of the molecule graph G is given. In this section, we describe a dynamic programming algorithm that solves the BOND TYPE ASSIGNMENT problem using the nice tree decomposition of the molecule graph G .

The tree T is rooted at an arbitrary bag. Above this root we add additional forget nodes, such that the new root contains a single vertex. Let X_r denote the new root of the tree decomposition and v_r denote the single vertex contained in

X_r . Analogously, we add additional introduce nodes under every leaf of T , such that the new leaf also contains a single vertex.

The vertices inside a bag X_i are referred to as $v_1, v_2, \dots, v_k$ where $k \leq \omega$. For simplicity of presentation, we assume that all edges $v_1 v_2, v_1 v_3, \dots, v_{k-1} v_k$ are present in each bag. Otherwise, the recurrences can be simplified accordingly.

Let Y_i denote the vertices in G that are contained in the bags of the subtree below bag X_i . We assign a score matrix D_i to each bag X_i of the tree decomposition: let $D_i[a_1, \dots, a_k; b_{1,2}, \dots, b_{k-1,k}]$ be the minimum score over all valency assignments to the vertices in $Y_i \setminus X_i$ if for every $l = 1, \dots, k$, a_l bonds of vertex v_l have been consumed by the vertices in $Y_i \setminus X_i$, and bond types $b_{1,2}, \dots, b_{k-1,k}$ are assigned to edges $v_1 v_2, v_1 v_3, \dots, v_{k-1} v_k$. Using this definition, we delay the scoring of any vertex to the forget node where it is removed from a bag. This is advantageous since every vertex except for the root vertex v_r is forgotten exactly once, and since the exact valence of a vertex is not known until it is forgotten in the tree decomposition. Finally, we can compute the minimum score among all assignments using the root bag $X_r = \{v_r\}$ as $\min_a s_{v_r}(a) + D_r[a]$.

Our algorithm begins at the leaves of the tree decomposition and computes the score matrix D_i for every node X_i when score matrices of its children nodes have been computed. We initialize the matrix D_j of each leaf $X_j = \{v\}$ with

$$D_j[a_1; \cdot] = \begin{cases} 0 & \text{if } a_1 = 0, \\ \infty & \text{otherwise.} \end{cases}$$

During the bottom-up travel, the algorithm distinguishes if X_i is a forget node, an introduce node, or a join node, and computes D_i as follows:

Introduce nodes. Let X_i be a parent node of X_j such that $X_j = \{v_1, \dots, v_{k-1}\}$ and $X_i = \{v_1, \dots, v_k\}$. Then

$$D_i[a_1, \dots, a_k; b_{1,2}, \dots, b_{k-1,k}] = \begin{cases} D_j[a_1, \dots, a_{k-1}; b_{1,2}, \dots, b_{k-2,k-1}] & \text{if } a_k = 0, \\ \infty & \text{otherwise.} \end{cases}$$

Forget nodes. Let X_i be a parent node of X_j such that $X_j = \{v_1, \dots, v_k\}$ and $X_i = \{v_1, \dots, v_{k-1}\}$. Then

$$D_i[a_1, \dots, a_{k-1}; b_{1,2}, \dots, b_{k-2,k-1}] = \min_{\substack{b_{1,k}, \dots, b_{k-1,k} \in \{0,1,2\} \\ a_k \in \{0, \dots, A_{v_k}^*\}}} \left\{ s_{v_k} \left(a_k + \sum_{l=1}^k b_{l,k} \right) \right. \\ \left. + D_j[a_1 - b_{1,k}, \dots, a_{k-1} - b_{k-1,k}, a_k; b_{1,2}, \dots, b_{k-1,k}] \right\}$$

Join nodes. Let X_i be a parent node of X_j and X_h such that $X_i = X_j = X_h$. Then

$$D_i[a_1, \dots, a_k; b_{1,2}, \dots, b_{k-1,k}] = \min_{\substack{a'_l = 0, \dots, a'_l \\ \text{for } l=1, \dots, k}} \left\{ D_j[a'_1, \dots, a'_k; b_{1,2}, \dots, b_{k-1,k}] + D_h[a_1 - a'_1, \dots, a_k - a'_k; b_{1,2}, \dots, b_{k-1,k}] \right\}$$

For simplicity of the presentation of our algorithm, we assumed above that every two vertices in each bag of the tree decomposition are connected by an edge, but in reality, the degree of a vertex in a molecule graph cannot exceed the maximum valence $d \leq 7$ of an atom in the molecule graph. Therefore, the number of edges in a bag is upper-bounded by ωd .

Lemma 1. *Given a nice tree decomposition of a molecule graph G , the algorithm described above computes an optimal assignment for the BOND TYPE ASSIGNMENT problem on G in time $O(a_{max}^{2\omega} \cdot 3^\alpha \cdot \omega \cdot m + m)$, where $a_{max} = \max_v A_v^*$ is the maximum (open) valence of an atom, m and $\omega - 1$ are size and width of the tree decomposition, d is the maximum degree in the molecule graph, and $\alpha := \min\{\binom{\omega}{2}, \omega d\}$.*

Due to space constraints, we defer the proof of Lemma 1 to the full paper. If the optimal solution is saved during the bottom-up processing, we can traverse the tree decomposition top-down afterwards to obtain all bond types of the molecule. This can be done in time $O(m)$.

5 Algorithm Engineering and Computational Results

Clearly, we do not have to compute or store entries $D_j[a_1, \dots, a_k; b_{1,2}, \dots, b_{k-1,k}]$ with $a_i + \sum_j b_{i,j} > A_i^*$ for some i , because such entries will never be used for the computation of minima in forget nodes or the root. We may implicitly assume that all such entries are set to infinity. Instead of an array, we use a hash map and store only those entries of D that are smaller than infinity. This reduces both running times and memory of our approach in applications.

To evaluate the performance of our algorithm, we implemented the algorithm in Java. All computations were done on an AMD Opteron-275 2.2 GHz with 6 GB of memory running Solaris 10. For our experiment, we used the MMFF94 dataset,¹ which consists of 760 molecule graphs predominantly derived from the Cambridge Structural Database. Bond types are given in the dataset but we removed this information and reassigned the bond types to those molecule graphs. We removed 30 molecule graphs that contain elements such as lithium or chlorine not covered in our scoring table (see below), or that have atom bindings such as oxygen atoms connected to three other atoms, that are also not covered in our scoring. The largest molecule graphs contains 59 atoms, the smallest 3 atoms, the average 23 atoms.

To compute the optimal tree decompositions of the molecule graphs, we used the method QuickBB in the library LibTW implemented by van Dijk *et al.* (<http://www.treewidth.com>). We implemented a method to transform the computed optimal tree decompositions into nice tree decompositions. In view of the near-outerplanarity of molecule graphs, we expected treewidths to be

¹ <http://www.ccl.net/cca/data/MMFF94/>, source file MMFF94_datative.mo12, of Feb. 5, 2009.

Table 1. Overview on the data used in our experiment. “Treewidth” gives the range of treewidths in this group, and “TD” and “DP” are average running times for tree decomposition and dynamic programming in milliseconds, respectively. “average # solutions” is the number of solutions our algorithm found on average.

instance size		number of instances	treewidth	average treewidth	running time		average # solutions
$ V $	$ E $				TD	DP	
3–10	2–11	63	1–2	1.2	4	5	1.4
11–20	10–22	206	1–3	1.8	6	36	1.2
21–30	20–33	328	1–3	2.0	6	68	1.3
31–40	30–43	125	1–3	2.0	8	76	1.3
41–50	40–53	5	1–2	1.8	9	87	1.4
51–59	53–61	3	2	2.0	5	234	1.0

rather small: In fact, we find that 18.6% of all molecules have treewidth one, 96.6% have treewidth ≤ 2 , and all molecules have treewidth at most three. The average treewidth is 1.85.

For scoring an assignment, we use the scoring table from Wang *et al.* [12]. This scoring allows atoms to have rather “exotic” valences, but gives an *atomic penalty score* (aps) to these rare valence states. As an example, carbon is allowed to take valence two (with aps 64), three (aps 32), four (aps 0), five (aps 32), or six (aps 64). In addition, different scores can be applied for the same element, depending on the local neighborhood: For example, carbon in a carboxylate group COO^- can take valence four (aps 32), five (aps 0), or six (aps 32). See Table 2 in [12] for details.

See Table 1 for computational results. Total running times are usually well below one second, and 56 ms on average. We were able to recover the correct assignment in all cases. In some cases, the algorithm found up to six optimal solutions because of symmetries and aromatic rings in the molecule graph.

6 Conclusion

We considered the problem of assigning bond types to a molecule graph and showed that the problem is NP-hard. Based on the tree decomposition concept, we introduced a dynamic programming algorithm with running time practically linear in the size of the molecule. In contrast to the previous heuristic and integer linear programming based algorithms, our algorithm is the first algorithm that computes exact solutions for the problem in a guaranteed running time. Furthermore, the running time of our algorithm depends strongly on the structure of the graph, but not on the size of the graph. We expect that the algorithm can be applied to solve the problem on large molecules if the molecule graph has small treewidth.

As a next step of algorithm design, we will do further algorithm engineering to improve the running time of our algorithm for practical uses. Furthermore, we want to evaluate the quality of solutions and the running time of our algorithm

against other algorithms. In particular, we want to verify that our algorithm finds more chemically or biologically relevant solutions than heuristic approaches.

Acknowledgment

We thank Anne Dehof and Andreas Hildebrandt for introducing us to the problem. Q.B.A. Bui gratefully acknowledges financial support from the DFG, research group “Parameterized Algorithms in Bioinformatics” (BO 1910/5).

References

1. Alber, J., Dorn, F., Niedermeier, R.: Experimental evaluation of a tree decomposition based algorithm for Vertex Cover on planar graphs. *Discrete Appl. Math.* 145(2), 219–231 (2005)
2. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embedding in a k -tree. *SIAM J. Algebra. Discr.* 8, 277–284 (1987)
3. Baber, J.C., Hodgkin, E.E.: Automatic assignment of chemical connectivity to organic molecules in the cambridge structural database. *J. Chem. Inf. Model.* 32, 401–406 (1992)
4. Bodlaender, H.L.: A partial k -arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.* 209, 1–45 (1998)
5. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)
6. Froeyen, M., Herdewijn, P.: Correct bond order assignment in a molecular framework using integer linear programming with application to molecules where only non-hydrogen atom coordinates are available. *J. Chem. Inf. Model.* 5, 1267–1274 (2005)
7. Garey, M.R., Johnson, D.S.: *Computers and Intractability (A Guide to Theory of NP-Completeness)*. Freeman, New York (1979)
8. Hendlich, M., Rippmann, F., Barnickel, G.: BALI: automatic assignment of bond and atom types for protein ligands in the brookhaven protein databank. *J. Chem. Inf. Model.* 37, 774–778 (1997)
9. Kloks, T.: *Treewidth, Computation and Approximation*. Springer, Heidelberg (1994)
10. Niedermeier, R.: *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, Oxford (2006)
11. Robertson, N., Seymour, P.: Graph minors: algorithmic aspects of tree-width. *J. Algorithms* 7, 309–322 (1986)
12. Wang, J., Wang, W., Kollmann, P.A., Case, D.A.: Automatic atom type and bond type perception in molecular mechanical calculations. *J. Mol. Graph. Model.* 25, 247–260 (2006)

On the Diaconis-Gangolli Markov Chain for Sampling Contingency Tables with Cell-Bounded Entries

Ivona Bezáková¹, Nayantara Bhatnagar^{2,*}, and Dana Randall^{3,**}

¹ Dept. of Computer Science, Rochester Institute of Technology, Rochester, NY, USA

² Dept. of Statistics, University of California, Berkeley, CA, USA

³ Dept. of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

Abstract. The problems of uniformly sampling and approximately counting contingency tables have been widely studied, but efficient solutions are only known in special cases. One appealing approach is the Diaconis and Gangolli Markov chain which updates the entries of a random 2×2 submatrix. This chain is known to be rapidly mixing for cell-bounded tables only when the cell bounds are all 1 and the row and column sums are regular. We demonstrate that the chain can require exponential time to mix in the cell-bounded case, even if we restrict to instances for which the state space is connected. Moreover, we show the chain can be slowly mixing even if we restrict to natural classes of problem instances, including regular instances with cell bounds of either 0 or 1 everywhere, and dense instances where at least a linear number of cells in each row or column have non-zero cell-bounds.

1 Introduction

We consider a popular approach for sampling standard and cell-bounded contingency tables based on a local Markov chain. Standard contingency tables are non-negative matrices consistent with prescribed row and column sums, while *cell-bounded* contingency tables additionally satisfy given cell bounds. More precisely, suppose that we are given a list of positive integers $r = (r_1, \dots, r_m)$ called the *row sums* and another list of positive integers $c = (c_1, \dots, c_n)$ called the *column sums* such that $\sum_{i=1}^m r_i = \sum_{j=1}^n c_j$. The set of *standard contingency tables* $\Sigma_{r,c}$ is the set of $m \times n$ non-negative integer matrices T that satisfy the given row and column sums. In the more general case of *cell-bounded contingency tables*, we are given row sums r , column sums c , and *cell-bounds* b_{ij} with $1 \leq i \leq m$ and $1 \leq j \leq n$ satisfying $b_{ij} \leq \min(r_i, c_j)$, for all i, j . Then the cell-bounded contingency tables $\Sigma_{r,c,b}$ are the $m \times n$ non-negative matrices T such that $\sum_{j=1}^n t_{i,j} = r_i$, $\sum_{i=1}^m t_{i,j} = c_j$, and $0 \leq t_{i,j} \leq b_{i,j}$ for all i, j .

The problem of sampling contingency tables almost uniformly at random was first studied for its applications in statistics [5]. Subsequently, there has been a

* Supported by DOD ONR grant N0014-07-1-05-06 and DMS-0528488.

** Supported in part by NSF grants CCF-0830367 and DMS-0505505.

long line of work focused on sampling and approximately counting contingency tables (see, e.g., [1,2,3,7,8,12,15]). Despite this extensive amount of work, these problems have been solved only in special cases. Dyer, Kannan and Mount [8] and Morris [14] showed that there is an *fpras*, a fully polynomial randomized approximation scheme (see [16]), for approximately counting standard contingency tables when the row and column sums are all sufficiently large using methods for estimating the volume of a convex body. There are also a variety of proofs showing there is an *fpras* for approximately counting standard tables when there are only a constant number of rows: by Cryan and Dyer [2] using dynamic programming and volume estimation, by Cryan et al. [3] using a local Markov chain, and by Dyer [7] using dynamic programming. Cryan, Dyer and Randall [4] extended these results to the more general setting with cell bounds by showing that there is also an *fpras* for cell-bounded tables in the same two cases: when the row and column sums are large or when the number of rows is a constant. Their proofs used dynamic programming and volume estimation. In addition to generalizing previous results, this work is significant because cell-bounded tables are *self-reducible*, the class of problems for which there is a known reduction between approximate counting and sampling (see [11]), so the existence of an *fpras* implies we can efficiently sample cell-bounded tables in these cases as well. The standard contingency table problem is not known to have this useful property.

Our work here focuses specifically on the Markov chain approach introduced by Diaconis and Gangolli [6]. In each step of the chain we choose two rows and two columns at random, and then we modify the entries in this 2×2 submatrix by trying to add a fixed integer to two diagonal entries and subtracting the same amount from the other two entries, so long as the resulting matrix is non-negative (and still satisfies the cell-bounds, if they exist). In the case of standard contingency tables, the Diaconis-Gangolli chain always connects the state space and is known to be rapidly mixing when there are a constant number of rows [2]. However, there is no evidence suggesting that this chain is a poor approach to sampling in other situations.

Even less is known for cell-bounded tables. Kannan, Tetali and Vempala [12] showed that the Diaconis-Gangolli chain is rapidly mixing when the cell bounds are all 1 and the row and column sums are regular. However, if the cell bounds are restricted to being either 0 or 1, the chain might not even connect the state space. Consider, for example, the 3×3 case where $r = (1, 1, 1)$, $c = (1, 1, 1)$ and $b_{1,1} = b_{2,2} = b_{3,3} = 0$; note that it is not possible to move between the two valid tables with moves of the Markov chain.

It is natural to ask whether the Diaconis-Gangolli chain is always rapidly mixing when restricted to instances $\Sigma_{r,c,b}$ where the state space is connected. We show that in the cell-bounded setting, even when the chain connects the space, it can require exponential time to converge. While not so surprising, no such evidence exists for standard tables. This may help explain why there has been such limited success showing the chain is fast for large classes of graphs.

Further, we give additional evidence that this chain can be a bad approach to sampling cell-bounded contingency tables by considering two natural special

cases of inputs. We show that the chain can require exponential time even in the dense case (the number of cells with positive cell-bounds in each row and column is linear) and the regular case (when all the r_i and c_j are equal). Our proofs of slow mixing are based on demonstrating a bottleneck in the state space that implies that the *conductance* of the Markov chain is very small.

2 Preliminaries

Let $m, n \in \mathbf{N}$ and let $r = (r_1, r_2, \dots, r_m) \in \mathbf{N}^m$ and $c = (c_1, c_2, \dots, c_n) \in \mathbf{N}^n$ be vectors with $\sum_{i=1}^m r_i = \sum_{j=1}^n c_j$. Let $b = b_{i,j} \in \mathbf{N}$ for every $i \in [m], j \in [n]$. A non-negative integer matrix $(t_{i,j})_{m \times n}$ is a *cell-bounded contingency table* with row sums r , column sums c , and cell-bounds b , if it satisfies:

- (i) for every $i \in [m]$, we have $\sum_{j=1}^n t_{i,j} = r_i$,
- (ii) for every $j \in [n]$, we have $\sum_{i=1}^m t_{i,j} = c_j$, and
- (iii) for every $i \in [m], j \in [n]$, we have $0 \leq t_{i,j} \leq b_{i,j}$.

The row and column sums are also called the *marginals*. Denote the set of all such tables by $\Sigma_{r,c,b}$. Note that no generality is lost by setting all the lower bounds on $t_{i,j}$ to be 0, since any lower bound $0 < a_{i,j} \leq b_{i,j}$ can be eliminated by replacing the cell-bound by $b_{i,j} - a_{i,j}$, and making the corresponding row and column sums $r_i - a_{i,j}$ and $c_j - a_{i,j}$, respectively.

The *cell-bounded contingency table problem* is to generate a (uniformly) random table in $\Sigma_{r,c,b}$. We describe below a well-known Markov chain which performs a random walk on the space of cell-bounded tables.

The Diaconis-Gangolli (DG) Markov chain

Given $T \in \Sigma_{r,c,b}$, the Markov chain performs a step as follows:

1. With probability $1/2$ do nothing, i. e., let $T' = T$.
2. Otherwise, choose $i_1 < i_2$ uniformly at random from $[m]$ and $j_1 < j_2$ uniformly at random from $[n]$.
3. With probability $1/2$ let $d = 1$, else let $d = -1$.
4. Let $D = (d_{i,j})_{m \times n}$ be a matrix with only four non-zero entries: $d_{i_1,j_1} = d_{i_2,j_2} = d$ and $d_{i_1,j_2} = d_{i_2,j_1} = -d$. If $T + D$ satisfies the cell bounds $b_{i,j}$, then let $T' = T + D$.
5. Return T' as the new contingency table.

Since this Markov chain is symmetric, its stationary distribution π is uniform over $\Sigma_{r,c,b}$. However, it is possible that for certain cell-bounds the Markov chain is not irreducible. In this work we focus on input instances for which the Markov chain is irreducible (and therefore also ergodic).

It is fairly standard to use the mixing time to bound the convergence time of a Markov chain. Let P denote the transition matrix of a Markov chain M with state space Ω . Thus, $P^t(x, \cdot)$ denotes the distribution after t steps of the chain, with starting state x . The *mixing time* $\tau_x(\varepsilon)$ of M starting at state $x \in \Omega$ is

$$\tau_x(\varepsilon) = \min\{t \geq 0 \mid d_{tv}(P^t(x, \cdot), \pi) \leq \varepsilon\},$$

where $d_{tv}(\mu, \nu) = \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|$ is the *total variation distance*. The mixing time of M is $\tau(\varepsilon) = \max_{x \in \Omega} \tau_x(\varepsilon)$.

The *conductance* Φ_M of an ergodic Markov chain $M = (\Omega, P)$ with stationary distribution π is defined as follows:

$$\Phi_M = \min_{S \subseteq \Omega, \pi(S) \leq 1/2} \frac{\sum_{s_1 \in S, s_2 \in \Omega \setminus S} \pi(s_1) P(s_1, s_2)}{\pi(S)}$$

The following theorem relates conductance to the mixing time.

Theorem 1 ([9,13]). *Let M be a Markov chain on Ω such that $M(u, u) \geq \frac{1}{2}$ for every $u \in \Omega$ and let $\pi_{\min} = \min_{x \in \Omega} \pi(x)$. Then,*

$$\frac{1}{2} \left(\frac{1}{2\Phi_M} - 1 \right) \log \left(\frac{1}{2\varepsilon} \right) \leq \tau(\varepsilon) \leq \frac{2}{\Phi_M^2} \log \left(\frac{1}{\pi_{\min}\varepsilon} \right).$$

3 Slow Mixing of the Diaconis-Gangolli Chain

Our main theorems demonstrate that the Diaconis-Gangolli chain has exponentially large mixing time in the cell-bounded case. Before stating our results, it will be useful to present a graph interpretation of cell-bounded tables when all the cell bounds $b_{i,j}$ are 0 or 1. Let $G = (V_1, V_2, E)$ be a bipartite graph with partition sizes m, n and adjacency matrix $B = \{b_{i,j}\}$, i. e., $V_1 = \{v_{1,1}, \dots, v_{m,1}\}$, $V_2 = \{v_{1,2}, \dots, v_{n,2}\}$, and $E = \{(v_{i,1}, v_{j,2}) \mid b_{i,j} > 0\}$. Then a cell-bounded contingency table T satisfying row sums r_i , column sums c_j and bounds $b_{i,j}$ can be interpreted as a subgraph of G with degree requirements $\deg_T(v_{i,1}) = r_i$ for every $i \in [m]$ and $\deg_T(v_{j,2}) = c_j$ for every $j \in [n]$.

3.1 Dense Instances

We show there is a family of *dense* instances, i.e., where at least a linear number of cell bounds in each row and column are non-zero, for which the Diaconis-Gangolli chain connects the space but mixes exponentially slowly. Dense instances are significant because they are close to the graphical case with all $b_{i,j} = 1$.

Theorem 2. *There exists a family of $n \times n$ instances of the cell-bounded contingency table problem such that*

- (i) *the upper-bound on each cell is either 0 or 1,*
- (ii) *the Diaconis-Gangolli chain connects the state space,*
- (iii) *the number of cells with non-zero upper-bound is at least $n/4$ in each row and each column, and*
- (iv) *the mixing time of the chain is greater than an exponential in n .*

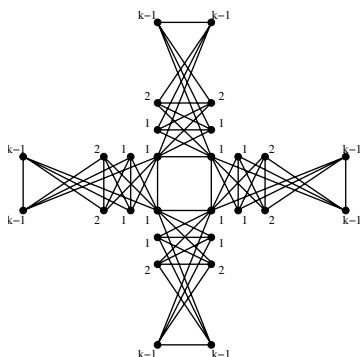


Fig. 1. The graph representation of the input instances from Theorem 2

Proof. Since all the upper-bounds for all cells are at most 1, we can use the graph representation of the problem. Before we describe the family of $n \times n$ instances, we introduce an important building block for each instance, a complete bipartite graph on $k + k$ vertices: $H_k = (W_{k,1}, W_{k,2}, F_k)$ where $W_{k,\ell} = \{w_{1,\ell}, w_{2,\ell}, \dots, w_{k,\ell}\}$ for $\ell \in \{1, 2\}$, and $F_k = W_{k,1} \times W_{k,2}$.

An $n \times n$ instance G_n is a bipartite graph consisting of four copies of H_k , let us call them $H_k^{(1)}$, $H_k^{(2)}$, $H_k^{(3)}$, and $H_k^{(4)}$, where all the vertices are distinct except for $w_{1,2}^{(j)} = w_{1,1}^{(j+1)}$ for $j \in [3]$ and $w_{1,2}^{(4)} = w_{1,1}^{(1)}$ (and the vertices $w_{1,1}^{(j)}$, $j \in [4]$, are four distinct vertices), see Figure 1. We also need to specify the row and column sums, or, alternatively for the graph representation, the required degrees (see Figure 1): for $\ell \in [2]$ and $j \in [4]$,

$$\deg(w_{i,\ell}^{(j)}) = \begin{cases} i - 1 & \text{for } i > 1, \\ 1 & \text{for } i = 1. \end{cases}$$

Notice that part (i) of the theorem follows from the construction and since $n = 4k - 2$, part (iii) is also immediate. Before we prove the remaining two parts, let us show the following claim:

Claim. Any subgraph G' of G_n that satisfies the required degree sequence is exactly one of the following two types:

- a) both $w_{1,1}^{(1)}$ and $w_{1,2}^{(1)}$ are connected to vertices in $H_k^{(1)}$ in G' and both $w_{1,1}^{(3)}$ and $w_{1,2}^{(3)}$ are connected to vertices in $H_k^{(3)}$ in G' , or
- b) both $w_{1,1}^{(2)}$ and $w_{1,2}^{(2)}$ are connected to vertices in $H_k^{(2)}$ in G' and both $w_{1,1}^{(4)}$ and $w_{1,2}^{(4)}$ are connected to vertices in $H_k^{(4)}$ in G' .

Proof. We know that $w_{1,1}^{(1)} = w_{1,2}^{(4)}$ has exactly one neighbor in G' and by the definition of G_n , this neighbor is either in $H_k^{(1)}$ or $H_k^{(4)}$. Suppose first that the neighbor is in $H_k^{(1)}$. Then we claim that the (only) neighbor of $w_{1,2}^{(1)}$ must also

be in $H_k^{(1)}$. Suppose for contradiction that $w_{1,2}^{(1)}$ is connected to a vertex outside of $H_k^{(1)}$ (i. e., it is connected to a vertex in $H_k^{(2)}$). Let us consider the part $G'^{(1)}$ of G' that is a subgraph of $H_k^{(1)}$. Then the degrees (in $G'^{(1)}$) of the vertices in $W_{k,1}^{(1)}$ are $(1, 1, 2, \dots, k-1)$ whereas the degrees of the vertices in $W_{k,2}^{(1)}$ are $(0, 1, 2, \dots, k-1)$. However, this is not possible since the sum of the degrees of the vertices in $W_{k,1}^{(1)}$ must be equal to the sum of the degrees of the vertices in $W_{k,2}^{(1)}$ (and it equals the number of edges of $G'^{(1)}$). Thus, $w_{1,1}^{(1)}$ is connected to a vertex in $H_k^{(1)}$ if and only if $w_{1,2}^{(1)}$ is connected to a vertex in $H_k^{(1)}$. The statement of the claim follows by symmetry of $H_k^{(1)}, \dots, H_k^{(4)}$.

Part (ii) of the theorem follows from a proof by Kannan et al. [12] who show that the Diaconis-Gangolli chain connects the state space for every input instance consisting of the complete graph and all cell-bounds equal to 1. In our case the graph is not the complete graph; however, the H_k 's are complete graphs. Now suppose we want to move from a graph G'_1 to a graph G'_2 using Diaconis-Gangolli (DG) moves. Without loss of generality assume that the vertex $w_{1,1}^{(1)}$ is connected to a vertex in $H_k^{(1)}$. Then, by the above claim, G'_1 satisfies a) while G'_2 satisfies either a), or b). We will consider both cases: if G'_2 satisfies b), then we will use DG moves to perform the following modifications:

1. Consider the part G''_1 of G'_1 that is a subset of $H_k^{(1)}$ and the part G''_2 of G'_2 that is a subset of $H_k^{(1)}$: notice that the vertices $w_{1,1}^{(1)}$ and $w_{1,2}^{(1)}$ have degree 0 in G''_2 but all the other degrees are equal. Let G_2^* be identical to G''_2 with the edge $(w_{1,1}^{(1)}, w_{1,2}^{(1)})$ included. We use Kannan et al. to modify G''_1 into G_2^* .
2. Analogously, modify the part of G'_1 that is a subset of $H_k^{(3)}$ into the part of G'_2 that is a subset of $H_k^{(3)}$, with edge $(w_{1,1}^{(3)}, w_{1,2}^{(3)})$ included.
3. Using a single DG move, swap edges $(w_{1,1}^{(1)}, w_{1,2}^{(1)})$ and $(w_{1,1}^{(3)}, w_{1,2}^{(3)})$ for $(w_{1,1}^{(2)}, w_{1,2}^{(2)})$ and $(w_{1,1}^{(4)}, w_{1,2}^{(4)})$.
4. Modify the part of G'_1 that is a subset of $H_k^{(2)}$, with edge $(w_{1,1}^{(2)}, w_{1,2}^{(2)})$ included, into the part of G'_2 that is a subset of $H_k^{(2)}$.
5. Finally, modify the part of G'_1 that is a subset of $H_k^{(4)}$, with edge $(w_{1,1}^{(4)}, w_{1,2}^{(4)})$ included, into the part of G'_2 that is a subset of $H_k^{(4)}$.

If G'_2 satisfies a), then the situation is even easier:

1. Using the result of Kannan et al., modify the part of G'_1 that is a subset of $H_k^{(1)}$ into the part of G'_2 that is subset of $H_k^{(1)}$.
2. Similarly, modify parts of G'_1, G'_2 that are subsets of $H_k^{(3)}$.
3. Notice that the part of G'_1 that is a subset of $H_k^{(2)} \setminus \{w_{1,1}^{(2)}, w_{1,2}^{(2)}\}$ is uniquely given (its degree sequence is $(1, 2, \dots, k-1), (1, 2, \dots, k-1)$ and there is a single graph satisfying this degree sequence); the same holds for the corresponding part of G'_2 . Thus, they are identical and no modification is necessary.

4. Again, no modification is necessary for the parts of G'_1, G'_2 that are subsets of $H_k^{(4)} \setminus \{w_{1,1}^{(4)}, w_{1,2}^{(4)}\}$.

It remains to prove part (iv) of the theorem which will follow from an upper bound on the conductance of the chain. Define the set S to be the set of graphs satisfying part a) of the above claim. We can define a bijection between S and $\Sigma_{r,c,b} \setminus S$ by symmetry, so $\pi(S) = 1/2$. We need to compute $\sum_{s_1 \in S, s_2 \in \Sigma_{r,c,b} \setminus S} \pi(s_1)P(s_1, s_2)$. Suppose that $s_1 \in S$ and there exists $s_2 \in \Sigma_{r,c,b} \setminus S$ such that $P(s_1, s_2) > 0$. Then, s_1 is a graph containing the edges $(w_{1,1}^{(1)}, w_{1,2}^{(1)})$ and $(w_{1,1}^{(3)}, w_{1,2}^{(3)})$. Notice there is a single graph satisfying this condition; moreover, there is a single $s_2 \in \Sigma_{r,c,b} \setminus S$ such that $P(s_1, s_2) > 0$ (more precisely, $P(s_1, s_2) = \frac{1}{n(n-1)^2}$). The last quantity to bound is $\pi(s_1)$.

Since $\pi(s_1) = 1/|\Sigma_{r,c,b}|$, we need to estimate $|\Sigma_{r,c,b}|$. Let $T(k)$ be the number of subgraphs of H_k with the degree sequence $(1, 1, 2, \dots, k-1), (1, 1, 2, \dots, k-1)$. It is not difficult to see that $T(k) \geq 4T(k-2)$ and $T(2) = 2, T(1) = 1$. (The inequality $T(k) \geq 4T(k-2)$ follows from the fact that the vertices with required degree $k-1$ are connected to all but one of their neighbors. Suppose that the omitted neighbors would be the vertices of required degree 1. Without loss of generality, let the omitted vertices be $w_{1,1}$ and $w_{1,2}$. After updating the required degree, we get that the new required degree sequence is $(0, 1, 1, 2, 3, \dots, k-3), (0, 1, 1, 2, 3, \dots, k-3)$. Thus, since we have two choices for the vertex of degree 1 for both W_1 and W_2 , we get that $T(k) \geq 2^2 T(k-2)$.) Therefore, $T(k) \geq 2^{k-1}$.

Now we can estimate $|\Sigma_{r,c,b}|$: if a $G' \in \Sigma_{r,c,b}$ satisfies part a) of the above claim, then the intersection of G' and $H_k^{(2)}$ is fixed, similarly the intersection of G' and $H_k^{(4)}$. The intersection of G' with $H_k^{(1)}$ is simply any subgraph of $H_k^{(1)}$ satisfying the degree sequence $(1, 1, 2, \dots, k-1), (1, 1, 2, \dots, k-1)$; similarly for the intersection of G' with $H_k^{(3)}$. Thus, the graphs satisfying part a) of the above claim contribute at least $(2^{k-1})^2 = 2^{n/2-1}$ to $|\Sigma_{r,c,b}|$. We get the same estimate for graphs satisfying part b), therefore $|\Sigma_{r,c,b}| \geq 2^{n/2}$.

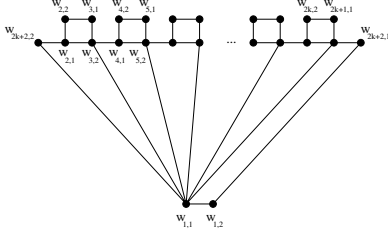
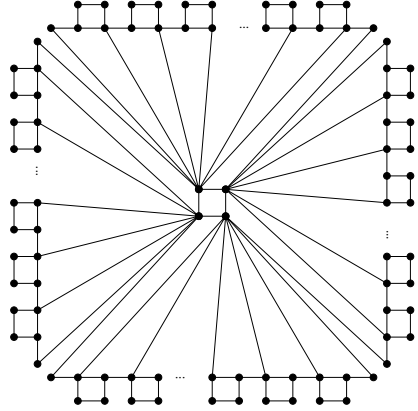
Finally, we bound the conductance as follows:

$$\Phi_M \leq \frac{\sum_{s \in S, t \notin S} \pi(s)P(s, t)}{\pi(S)} = \frac{\frac{1}{|\Sigma_{r,c,b}|} \cdot \frac{1}{n^2(n-1)^2}}{\frac{1}{2}} = \frac{2}{2^{n/2} \cdot n^2(n-1)^2} \leq \frac{1}{2^{n/2}}.$$

Therefore, part (iv) of the theorem follows from Theorem 1.

3.2 Regular Instances

Kannan, et al. [12] showed that if the cell-bounds are all 1, then the Diaconis-Gangolli chain mixes rapidly for all regular instances, i.e., the r_i 's and the c_j 's are all identical. What happens if the instance is regular but there are cell bounds? It is not difficult to find instances for which the Diaconis-Gangolli chain does not connect the state space. However, even in those cases when the state space is connected, is it always rapidly mixing? We answer this question negatively.

**Fig. 2.** The graph H_k from Theorem 3**Fig. 3.** The graph representation of the input instances from Theorem 3

Theorem 3. *There exists a family of $n \times n$ instances of the cell-bounded contingency table problem such that*

- (i) *the upper-bound on each cell is either 0 or 1,*
- (ii) *the Diaconis-Gangolli chain connects the state space,*
- (iii) *the marginals are regular with $r_i = 1$ and $c_j = 1$ for every $i, j \in [n]$, and*
- (iv) *the mixing time of the chain is at least exponential.*

Proof. Similarly to the instances from Theorem 2, the instances here consist of four identical building blocks. Each building block $H_k = (W_{k,1}, W_{k,2}, F_k)$ is a bipartite graph on $(2k+2) + (2k+2)$ vertices, where $W_{k,\ell} = \{w_{1,\ell}, w_{2,\ell}, \dots, w_{2k+2,\ell}\}$ for $\ell \in \{1, 2\}$, and F_k is defined as follows, see also Figure 2:

$$F_k = \{(w_{1,1}, w_{1,2}), (w_{1,1}, w_{2k+2,2}), (w_{2k+2,1}, w_{1,2}), (w_{2,1}, w_{2k+2,2})\} \cup \\ \{(w_{1,1}, w_{2i+1,2}) \mid i \in [k]\} \cup \{(w_{2(i+1),1}, w_{2i+1,2}) \mid i \in [k]\} \cup \\ \{(w_{2i,1}, w_{2i,2}), (w_{2i,1}, w_{2i+1,2}), (w_{2i+1,1}, w_{2i+1,2}), (w_{2i+1,1}, w_{2i,2}) \mid i \in [k]\}.$$

As before, the graph G_n consists of four copies of H_k , let's call them $H_k^{(j)}$, $j \in [4]$, where all the vertices are distinct except for $w_{1,2}^{(j)} = w_{1,1}^{(j+1)}$ for $j \in [3]$ and $w_{1,2}^{(4)} = w_{1,1}^{(1)}$. The construction is depicted on Figure 3.

Parts (i) and (iii) of the theorem follow immediately from the construction. We will now prove part (ii). First let us observe that Claim 3.1 from Theorem 2 holds for the graph G_n from this proof. We will prove (ii) by first showing that it is possible to use DG moves to move between any two subgraphs of G_n satisfying part a) of the claim (and, analogously, the same holds for two subgraphs satisfying part b)), and then we show that there is a subgraph satisfying part a) and a subgraph satisfying part b) such that it is possible to move between them.

We make one more observation: if G' is a subgraph of G_n satisfying part a) of the claim, then the intersection of G' with both $H_k^{(2)}$ and $H_k^{(4)}$ is uniquely defined.

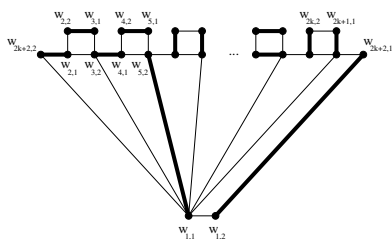


Fig. 4. A possible subgraph of the graph H_k with all-one degree sequence

This follows from the fact that there is a unique subgraph of H_k such that the vertices $w_{1,1}$ and $w_{1,2}$ have degree 0 while all other vertices have degree 1. (This subgraph contains exactly the edges $(w_{2i+1,1}, w_{2i,2})$ and $(w_{2i+2,1}, w_{2i+1,2})$ for every $i \in [k]$, and the edge $(w_{2,1}, w_{2k+2,2})$.)

Now we construct a subgraph G' satisfying part a) and a subgraph G'' satisfying part b) such that we can move from G' to G'' in a single DG move: let G' contain the edges $(w_{1,1}^{(1)}, w_{1,2}^{(1)})$ and $(w_{1,1}^{(3)}, w_{1,2}^{(3)})$ (notice that by the above argument all the other edges of G' are uniquely defined); similarly, let G'' contain the edges $(w_{1,1}^{(2)}, w_{1,2}^{(2)})$ and $(w_{1,1}^{(4)}, w_{1,2}^{(4)})$. It follows immediately that G' and G'' are connected by a single DG move.

Suppose we have two subgraphs G' and G'' both satisfying part a) of the claim. To show we can move from G' to G'' using DG moves, we first show we can move from G' to the subgraph of H_k with all-one degree sequence that includes the edge $(w_{1,1}, w_{1,2})$. Let us call this graph $\tilde{G}'$. Suppose G' does not include this edge (or we are done). Therefore, it must connect $w_{1,1}$ to some other vertex, let it be $w_{2i+1,2}$ where $i \in [k]$ (if $w_{1,1}$ is connected to $w_{2k+2,2}$, analogous arguments hold). Then, the part of G' involving vertices $w_{j,\ell}$ for $j \in [2i+1] \cup \{2k+2\}$ and $\ell \in [2]$ is uniquely determined, and the rest can be chosen from exactly 2^{k-i} possibilities (see Figure 4). We can do the following DG moves:

1. For j going from $i+1$ to k do:
2. If the graph G' contains the edges $(w_{2j,1}, w_{2j,2})$ and $(w_{2j+1,1}, w_{2j+1,2})$, swap them for $(w_{2j,1}, w_{2j+1,2})$ and $(w_{2j+1,1}, w_{2j,2})$.
3. Swap the edges $(w_{1,1}, w_{2j-1,2})$ and $(w_{2j,1}, w_{2j+1,2})$ for $(w_{2j,1}, w_{2j-1,2})$ and $(w_{1,1}, w_{2j+1,2})$.
4. Finally, swap the edges $(w_{1,1}, w_{2k+1,2})$ and $(w_{2k+2,1}, w_{1,2})$ for the edges $(w_{1,1}, w_{1,2})$ and $(w_{2k+2,1}, w_{2k+1,2})$.

We end up with the graph $\tilde{G}'$, as promised.

Last, we show part (iv) of the theorem. As in the proof of Theorem 2, we bound the conductance of the chain: let S be the set of all subgraphs of G_n that satisfy part a) of the claim. Then, $\pi(S) = 1/2$ and

$$\sum_{s_1 \in S, s_2 \in \Sigma_{r,c,b} \setminus S} \pi(s_1) P(s_1, s_2) = \frac{1}{|\Sigma_{r,c,b}|} \frac{1}{n^2(n-1)^2},$$

since there is a unique pair of graphs $s_1 \in S_1$, $s_2 \in S_2$ such that $P(s_1, s_2) > 0$, namely $P(s_1, s_2) = \frac{1}{4(n(n-1)/2)^2}$. To estimate $|\Sigma_{r,c,b}|$, recall that there are exactly 2^{k-i} subgraphs of H_k that satisfy the all-one degree sequence and contain the edge $(w_{1,1}, w_{2i+1,2})$. Therefore, there are $\sum_{i=1}^k 2^{k-i} + 2^k + 1 = 2^{k+1}$ subgraphs of H_k with all-one degree sequence (2^k of those that contain the edge $(w_{1,1}, w_{2k+2,2})$ and 1 that contains the edge $(w_{1,1}, w_{1,2})$). Thus, $|\Sigma_{r,c,b}| = 2(2^{k+1})^2 = 2^{2k+3} = 2^{n/4+3/2}$ and the conductance is bounded by $\frac{1}{n^2(n-1)^2 2^{n/4}}$, an inverse exponential function. This concludes the proof of part (iv) of the theorem.

References

1. Bezáková, I., Bhatnagar, N., Vigoda, E.: Sampling binary contingency tables with a greedy start. *Random Structures and Algorithms* 30, 168–205 (2007)
2. Cryan, M., Dyer, M.: A polynomial-time algorithm to approximately count contingency tables when the number of rows is constant. *Journal of Computer and System Sciences* 67, 291–310 (2003)
3. Cryan, M., Dyer, M., Goldberg, L., Jerrum, M., Martin, R.: Rapidly mixing Markov chains for sampling contingency tables with a constant number of rows. In: *Proc. 43rd IEEE Symposium on Foundations of Computer Science*, pp. 711–720 (2002)
4. Cryan, M., Dyer, M., Randall, D.: Approximately counting integral flows and cell-bounded contingency tables. In: *Proc. 37th ACM Symposium on Theory of Computing*, pp. 413–422 (2005)
5. Diaconis, P., Efron, B.: Testing for independence in a two-way table: new interpretations of the chi-square statistic. *Annals of Statistics* 13, 845–913 (1995)
6. Diaconis, P., Gangolli, A.: Rectangular Arrays with Fixed Margins. In: Aldous, D., et al. (eds.) *Discrete Probability and Algorithms*, pp. 15–41. Springer, Heidelberg (1995)
7. Dyer, M.: Approximate counting by dynamic programming. In: *Proc. 35th ACM Symposium on the Theory of Computing*, pp. 693–699 (2003)
8. Dyer, M., Kannan, R., Mount, J.: Sampling contingency tables. *Random Structures & Algorithms* 10, 487–506 (1997)
9. Jerrum, M., Sinclair, A.: Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation* 82, 93–133 (1989)
10. Jerrum, M., Sinclair, A., Vigoda, E.: A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *Jour. ACM* 51, 671–697 (2004)
11. Jerrum, M., Valiant, L., Vazirani, V.: Random generation of combinatorial structures from a uniform distribution. *Theoretical Comp. Sci.* 43, 169–188 (1986)
12. Kannan, R., Tetali, P., Vempala, S.: Simple Markov-chain algorithms for generating bipartite graphs and tournaments. *Random Structures and Algorithms* 14, 293–308 (1999)
13. Lawler, G., Sokal, A.: Bounds on the L2 spectrum for Markov chains and Markov processes: a generalization of Cheeger’s inequality. *Trans. Amer. Math. Soc.* 309, 557–580 (1988)
14. Morris, B.: Improved bounds for sampling contingency tables. *Random Structures & Algorithms* 21, 135–146 (2002)
15. Mount, J.: Application of convex sampling to optimization and contingency table generation, PhD thesis, Carnegie Mellon University (1995); Technical Report CMU-CS-95-152, Department of Computer Science
16. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley, Reading (1994)

Finding a Level Ideal of a Poset

Shuji Kijima¹ and Toshio Nemoto²

¹ Research Institute for Mathematical Sciences,
Kyoto University, Kyoto, 606-8502, Japan

kijima@kurims.kyoto-u.ac.jp

² Graduate School of Information and Communication,
Bunkyo University, Kanagawa, 253-8550, Japan

nemoto@shonan.bunkyo.ac.jp

Abstract. This paper is concerned with finding a *level ideal* (LI) of a partially ordered set (poset): given a finite poset P , a level of each element $p \in P$ is defined as the number of ideals which do not include p , then the problem is to find an ideal consisting of elements whose levels are less than a given integer i . We call the ideal as the i -th LI. The concept of the level ideal is naturally derived from the *generalized median stable matching*, that is a fair stable marriage introduced by Teo and Sethuraman (1998). Cheng (2008) showed that finding the i -th LI is $\#P$ -hard when $i = \Theta(N)$, where N is the total number of ideals of P . This paper shows that finding the i -th LI is $\#P$ -hard even if $i = \Theta(N^{1/c})$ where $c \geq 1$ is an arbitrary constant. Meanwhile, we give a polynomial time exact algorithm when $i = O((\log N)^{c'})$ where c' is an arbitrary positive constant. We also devise two *randomized approximation schemes* using an oracle of almost uniform sampler for ideals of a poset.

1 Introduction

Let P be a finite partially ordered set (poset) with a partial order $\preceq$. A set $X \subseteq P$ is an *ideal*¹ of P if, whenever $x \in X$ and $y \preceq x$, we have $y \in X$ (see e.g., [4]). Let $\mathcal{D}(P)$ denote the set of ideals of P , and let $N = |\mathcal{D}(P)|$. We define a (level set) function g on P by

$$g(x) \stackrel{\text{def.}}{=} |\{X \in \mathcal{D}(P) \mid x \notin X\}| \quad (x \in P), \quad (1)$$

and define a (sublevel) set $S_i \subseteq P$ for $i \in \{1, \dots, N\}$ by

$$S_i \stackrel{\text{def.}}{=} \{x \in P \mid g(x) < i\}. \quad (2)$$

Note that every S_i is an ideal of P , since the function g is monotone increasing w.r.t. $\preceq$, meaning that if $x \preceq y$ then $g(x) \leq g(y)$. We call the ideal S_i as the (i -th) *level ideal* (or *LI*, for short). Figure 1 shows an example of a poset P , its level set function g , $\mathcal{D}(P)$, and level ideals².

¹ a.k.a. an *order ideal* or a (*closed*) *down set*. In addition, ideals and *antichains* are bijective.

² $\mathcal{F}(P)$ in Figure 1 denotes the set of level ideals, defined by (3) appearing later.

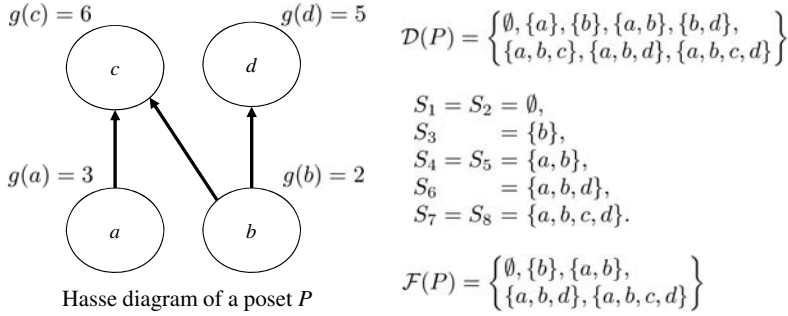


Fig. 1. An example of level ideals of a poset P

This paper is concerned with finding the i -th LI. The problem is naturally derived from the *generalized median stable matching (GMSM)* problem introduced by Teo and Sethuraman [23] in a context of a fairness of the stable marriage.

Background. The stable marriage problem is introduced by Gale and Shapley [6]: A problem instance is given by sets n -men and n -women, and lists of each person's preference order over the opposite sex. A *matching* is n pairs of a man and a woman, in which every person appears exactly once. For a matching, a pair of a man and a woman not in the matching is called *blocking pair* if they prefer each other to the current partner of each. A matching is *stable* unless a blocking pair exists. Gale and Shapley [6] showed that any instance has a stable matching by giving an algorithm to find it.

An instance of the stable marriage usually has a number of stable matchings. Furthermore, the preference of stable matchings is completely conflict between men and women; for any pair of stable matchings μ_1 and μ_2 , if all men prefer μ_1 to μ_2 , i.e., the partner in μ_1 is preferable or the same to the partner in μ_2 for each man, then all women prefer μ_2 to μ_1 , and vice versa. A fairness of a stable matching between men and women is a central issue of the stable marriage problem [17]. Teo and Sethuraman [23] devised the *GMSM* as a fair stable matching, and a number of papers followed after it [3,11,12,14,18,19]. Here we omit the details including the definition of the *GMSM*.

A strong connection between the stable marriage and ideals of a poset is well known. Conway pointed out that the set of stable matchings forms a distributive lattice w.r.t. the preference order of all men (and all women in reverse) [13]. Blair [2] showed that any finite distributive lattice has an instance of the stable marriage problem whose stable matchings is isomorphic to the given distributive lattice. Meanwhile, the set of ideals of a finite poset forms a distributive lattice, and Birkhoff's representation theorem says any finite distributive lattice is represented by the set of ideals of a poset (see e.g., [4]). In the stable marriage, the poset representing the stable matchings consists of "rotations" of partners,

and called the *rotation poset* [7]. The rotation poset for a stable marriage instance is found in $O(n^2)$ time, and a stable matching corresponding to an ideal of the rotation poset is obtained in $O(n^2)$ time. Conversely, as given a finite poset P , an instance of the stable marriage is constructed in $O(|P|^2)$ time [7].

Cheng [3] gave the first result on NP-hardness of finding the GMSM. In [3], she gave a characterization of GMSMs on the rotation poset, which had been independently discovered by Nemoto [14], and transformed the problem to finding a level ideal of a poset, then she showed that finding the i -th LI exactly is $\#P$ -hard when $i = \Theta(N)$, where N is the total number of ideals. Cheng also gave a simple exact algorithm for finding the i -th LI when $i = O(\log \log N)$, precisely when $i = O(\log |P|)$, and the complexity in case that i is $o(N)$ and $\omega(\log \log N)$ was left as open. Note that it remains to be seen whether or not the decision version of finding the i -th LI is in NP.

Results. We show that finding the i -th LI is $\#P$ -hard even when $i = \Theta(N^{1/c})$ for an arbitrary constant $c \geq 1$. We also show that even the query if a given ideal can be an i -th LI for some appropriate i is $\#P$ -hard. Meanwhile, we give an exact algorithm for the case of $i = O((\log N)^{c'})$ for an arbitrary positive constant c' . We devise two randomized approximation schemes for finding the i -th LI using an oracle for almost uniformly sampling ideals of a poset. For detail of some proofs, we refer to the preprint version [10].

Related works. Irving and Leather [8] showed that counting stable matchings is $\#P$ -complete by a reduction from counting antichains (or ideals) of a poset whose $\#P$ -hardness is due to Provan and Ball [16]. Steiner [22] gave a polynomial time algorithm based on dynamic programming for counting ideals of a poset of some special classes such as series-parallel, bounded width, etc. Propp and Wilson [15] devised a perfect sampler for ideals of a poset based on the monotone coupling from the past algorithm, whereas its expected running time becomes exponential in the size of the poset in the worst case. The existence of a polynomial time almost uniform sampler for ideals of a poset, or an FPRAS for counting, remains as a challenging problem [1]. Dubhashi et al. [5] discussed the relationship between finding a central element and counting ideals in a poset.

Organization. In Section 2, we establish two hardness results on the problems. In section 3, we give an exact algorithm in case of $i = O((\log N)^{c'})$. In Sections 4 and 5, we give two randomized approximation schemes for finding the i -th LI. Precisely, we in Section 4 give a simple RAS for the case of $i = O(N)$, while we in Section 5 give another artificial algorithm for general cases, which is effective especially in case of $i = o(N)$.

We denote the set of real numbers (non-negative, positive real numbers) by $\mathbb{R}$ ($\mathbb{R}_+$, $\mathbb{R}_{++}$), and the set of integers (non-negative, positive integers) by $\mathbb{Z}$ ($\mathbb{Z}_+$, $\mathbb{Z}_{++}$), respectively.

2 Hardness of Finding a Level Ideal

Let $\mathcal{F}(P) \subseteq \mathcal{D}(P)$ denote the family of (level) ideals

$$\mathcal{F}(P) \stackrel{\text{def.}}{=} \{S \subseteq P \mid S = S_i \ (i \in \{1, \dots, N\})\}. \quad (3)$$

We consider the following problem.

Problem 1. Given a poset P and an ideal $S \in \mathcal{D}(P)$, whether or not $S \in \mathcal{F}(P)$?

Theorem 1. *Problem 1 is $\#P$ -hard.*

To prove Theorem 1, we introduce three useful lemmas. Let P and Q be (disjoint) posets. The *disjoint union* $P \dot{\cup} Q$ is defined as follows; $x, y \in P \dot{\cup} Q$ satisfies $x \preceq y$ iff either $[x, y \in P \text{ and } x \preceq y]$ or $[x, y \in Q \text{ and } x \preceq y]$. The *linear sum* $P \oplus Q$ is defined as follows; $x, y \in P \oplus Q$ satisfies $x \preceq y$ iff the cases of $[x, y \in P \text{ and } x \preceq y]$, $[x, y \in Q \text{ and } x \preceq y]$, or $[x \in P \text{ and } y \in Q]$. Note that $P \dot{\cup} Q = Q \dot{\cup} P$, but $P \oplus Q \neq Q \oplus P$. The followings are known.

Lemma 1. [22] *Let P and Q be disjoint posets, then $|\mathcal{D}(P \dot{\cup} Q)| = |\mathcal{D}(P)| \cdot |\mathcal{D}(Q)|$.*

Lemma 2. [22,3] *Let P and Q be disjoint posets, then $|\mathcal{D}(P \oplus Q)| = |\mathcal{D}(P)| + |\mathcal{D}(Q)| - 1$.*

Lemma 3. [3] *For any $K \in \mathbb{Z}_{++}$, a poset Q satisfying $|\mathcal{D}(Q)| = K$ is realized in $\text{Poly}(\log K)$ time and space.*

We also define a set $U(x) \subseteq P$ for $x \in P$ by

$$U(x) \stackrel{\text{def.}}{=} \{y \in P \mid y \succeq x\}, \quad (4)$$

for convenience. Then note that we have $g(x) = |\mathcal{D}(P \setminus U(x))|$.

Proof of Theorem 1. We give a reduction from COUNTING IDEALS, that is to compute $|\mathcal{D}(P)|$ for a given poset P . The problem is known to be $\#P$ -complete [16]. Precisely, we consider a problem that given a poset P and an integer $K \in \mathbb{Z}_{++}$, the query is whether or not $|\mathcal{D}(P)| < K$. If we have an oracle for the query, we can compute $|\mathcal{D}(P)|$ by a binary search of K s between 0 and $2^{|P|}$. We in the following give a reduction from the query if $|\mathcal{D}(P)| < K$ to Problem 1.

For the integer K , let Q be a poset satisfying $|\mathcal{D}(Q)| = K$. The poset Q is constructed in $\text{Poly}(\log K)$ time by Lemma 3. Let R be a poset defined by $R \stackrel{\text{def.}}{=} (\{x\} \oplus P) \dot{\cup} (\{y\} \oplus Q)$ (see Figure 2). Now we consider $g(r)$ for each $r \in R$, defined by (1),

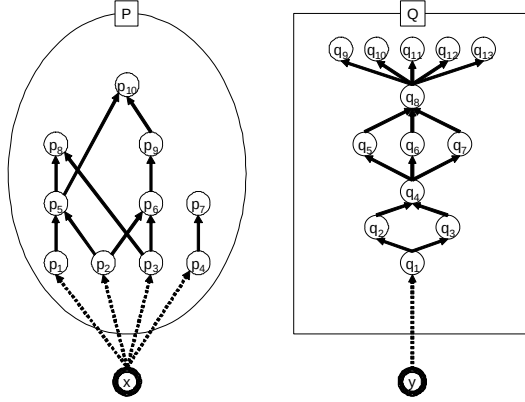


Fig. 2. An example of $R \stackrel{\text{def}}{=} (\{x\} \oplus P) \dot{\cup} (\{y\} \oplus Q)$

$$\begin{aligned}
 g(x) &= |\mathcal{D}(R \setminus U(x))| = |\mathcal{D}(\{y\} \oplus Q)| = 1 + |\mathcal{D}(Q)|, \\
 g(y) &= |\mathcal{D}(R \setminus U(y))| = |\mathcal{D}(\{x\} \oplus P)| = 1 + |\mathcal{D}(P)|, \\
 g(p) &= |\mathcal{D}(R \setminus U(p))| \geq |\mathcal{D}((\{y\} \oplus Q) \dot{\cup} \{x\})| = 2 \cdot g(x) \quad (\forall p \in P), \\
 g(q) &= |\mathcal{D}(R \setminus U(q))| \geq |\mathcal{D}((\{x\} \oplus P) \dot{\cup} \{y\})| = 2 \cdot g(y) \quad (\forall q \in Q),
 \end{aligned}$$

hold. With considering the definitions (2) and (3) of the set of level ideals $\mathcal{F}(R)$, we obtain the following three cases;

Case 1. If $|\mathcal{D}(P)| < |\mathcal{D}(Q)| = K$, then $\{x\} \notin \mathcal{F}(R)$ and $\{y\} \in \mathcal{F}(R)$, since $g(x) > g(y)$.

Case 2. If $|\mathcal{D}(P)| > |\mathcal{D}(Q)| = K$, then $\{x\} \in \mathcal{F}(R)$ and $\{y\} \notin \mathcal{F}(R)$, since $g(x) < g(y)$.

Case 3. Otherwise, i.e. $|\mathcal{D}(P)| = |\mathcal{D}(Q)| = K$, then $\{x\} \notin \mathcal{F}(R)$, $\{y\} \notin \mathcal{F}(R)$, and $\{x, y\} \in \mathcal{F}(R)$.

Thus, if we ask the oracle for Problem 1 whether $\{y\} \in \mathcal{F}(R)$, then ‘yes’ (Case 1) implies $|\mathcal{D}(P)| < K$ and ‘no’ (Cases 2 and 3) implies $|\mathcal{D}(P)| \geq K$. $\square$

Next we consider the following problem,

Problem 2. Given a poset P , an ideal $S \in \mathcal{D}(P)$, and a function $f : \mathbb{Z}_{++} \rightarrow \mathbb{Z}_{++}$, then let $i = f(|\mathcal{D}(P)|)$, and whether S is the i -th LI?

From Theorem 1, we observe that finding the i -th level ideal, that is Problem 2, is NP-hard even when $i = \Theta(\sqrt{N})$. Precisely, we obtain the following.

Proposition 1. *Given a poset R and an ideal $S \in \mathcal{D}(R)$, let $N = |\mathcal{D}(R)|$, then the problem whether or not S is the $\lceil \sqrt{N} \rceil$ -th LI of R is $\#P$ -hard. $\blacksquare$*

With some modifications to the proof of Proposition 1, we can establish a stronger result that finding the i -th level ideal, that is Problem 2, is $\#P$ -hard even when $i = \Theta(N^{1/c})$ for an arbitrary constant c ($c \geq 1$). See [10] for detail.

3 Exact Computation of the $\text{Poly}(|P|)$ -th LI

In this section, we consider the following problem,

Problem 3. Given a poset P and an integer $i \in \mathbb{Z}_{++}$, then find the i -th LI.

We give an exact algorithm for Problem 3, which runs in time in $O(i \cdot \text{Poly}(|P|))$. Thus the algorithm runs in time polynomial in the input size in case that i is $\text{Poly}(|P|)$, i.e., the case of $i = O((\log N)^c)$ for an arbitrary positive constant c . In other words, Problem 3 is solvable in time polynomial in the input size when $i = O((\log N)^c)$ for an arbitrary constant $c \geq 0$, since $N = |\mathcal{D}(P)|$ is at most $2^{|P|}$.

The algorithm is essentially based on (exhaustive) enumeration of ideals of a poset. Steiner [21] gave an enumeration algorithm for ideals of a poset, which generates all ideals one-by-one without duplication, and which runs in $O(|P|^2 + |P| \cdot |\mathcal{D}(P)|)$ time; more precisely the algorithm outputs every ideals in $O(|P|)$ time delay, after $O(|P|^2)$ time preprocessing. Squire [20] gave a faster algorithm running in $O(\log |P| \cdot |\mathcal{D}(P)|)$ time.

Now we describe the algorithm for Problem 3. Let $\mathcal{A}$ denote an enumeration algorithm of ideals of a poset. For each $p \in P$, we execute $\mathcal{A}$ for a poset $P \setminus U(p)$, and count up the number of ideals of $\mathcal{D}(P \setminus U(p))$ one-by-one. Let $Z(p)$ denote the value of a counter, then if $Z(p)$ reached at i we halt $\mathcal{A}$, and otherwise $\mathcal{A}$ stops with $Z(p) = |\mathcal{D}(P \setminus U(p))|$. Then $S = \{p \in P \mid Z(p) < i\}$ should be the i -th LI from the definition. See [10] for detail.

Clearly the time complexity of the algorithm is $O(|P| \cdot \mathcal{T}_{\mathcal{A}}(i))$, where $\mathcal{T}_{\mathcal{A}}(i)$ denotes the computation time in which enumeration algorithm outputs ideals up to i -th one, that is e.g., $O(|P|^2 + |P| \cdot i)$ by Steiner [21]. Thus it becomes a polynomial time algorithm when $i = \text{Poly}(|P|)$.

4 Simple Randomized Approximation for the i -th LI

In this section, we give a simple randomized approximation algorithm for the i -th LI. Theorem 1 suggests that finding just a level ideal in $\mathcal{F}(P)$ of a given poset P is $\#P$ -hard. Thus, we consider to find an ideal $S \in \mathcal{D}(P)$, which approximates the i -th level ideal S_i . We use the following oracle of almost uniform sampler on $\mathcal{D}(P)$ for a given poset P .

Oracle 1. (*Almost uniform sampler on ideals of a poset.*) Given an arbitrary ε ($0 < \varepsilon < 1$) and a poset P , Oracle returns an element of $\mathcal{D}(P)$ according to a distribution ν satisfying $d_{\text{TV}}(\pi, \nu) \stackrel{\text{def.}}{=} (1/2)\|\pi - \nu\|_1 \leq \varepsilon$, where π denotes the exactly uniform distribution on $\mathcal{D}(P)$.

Let γ_1 denote the time required for Oracle 1. Note that it is open whether γ_1 can be $\text{Poly}(|P|, -\ln \varepsilon)$. With using Oracle 1, we give the following simple randomized algorithm for Problem 2.

Algorithm 1. (ε -estimator for the λN -th LI.)

```

1 Input: A poset  $P$ ,  $\lambda$  ( $0 < \lambda < 1$ ),  $\varepsilon$  ( $0 < \varepsilon \leq \min\{\lambda, 1 - \lambda\}$ ),  $\delta$  ( $0 < \delta < 1$ ).
2 Set  $Z(p) := 0$  for each  $p \in P$ .
3 Repeat ( $T \stackrel{\text{def.}}{=} \lceil -12\varepsilon^{-2} \ln(\delta/|P|) \rceil$  times) {
4   Generate  $X \in \mathcal{D}(P)$  by Oracle 1 (where  $\nu$  satisfies  $d_{\text{TV}}(\pi, \nu) \leq \varepsilon/2$ ).
5   For(each  $p \in P$ ) {
6     if  $p \notin X$  then  $Z(p) := Z(p) + 1$ .
7   }
8 }
9 Set  $S \stackrel{\text{def.}}{=} \{p \in P \mid Z(p)/T < \lambda\}$ .
10 Output  $S$  and halt.
```

Theorem 2. *Algorithm 1 outputs an ideal $S \in \mathcal{D}(P)$ in $O((\gamma_1 + |P|)\varepsilon^{-2} \ln |P|/\delta)$ time, and S satisfies*

$$\Pr [S_{\lfloor (\lambda - \varepsilon)N \rfloor} \subseteq S \subseteq S_{\lceil (\lambda + \varepsilon)N \rceil}] \geq 1 - \delta. \quad (5)$$

Sketch of proof. The time complexity is easy to see. To show that the output S of Algorithm 1 is an ideal of P , it is enough to see that if a pair $p \in P$ and $q \in P$ satisfies $p \prec q$ and $q \in S$ then $p \in S$. For any random sample $X \in \mathcal{D}(P)$ in Step 1, if $q \in X$ then $p \in X$, since X is an ideal of P . It implies $Z(p) \geq Z(q)$ in Step 1, hence if $q \in S$ then $p \in S$ from the definition of S in Step 2.

To obtain the inequality (5), we show the following claim;

Claim. For any $p \in P$,

Case 1. if $g(p) \leq (\lambda - \varepsilon)N$, then the probability $p \notin S$ (i.e., $Z(p)/T \geq \lambda$) is at most $\delta/|P|$, and

Case 2. if $g(p) \geq (\lambda + \varepsilon)N$, then the probability $p \in S$ (i.e., $Z(p)/T < \lambda$) is at most $\delta/|P|$.

The claim can be proven by the Chernoff bound, but here we omit the detail. With using the claim, we can obtain the inequality (5). See [10] for detail. ■

5 Approximation for the i -th LI Based on Counting Ideals

The time complexity of Algorithm 1, in the previous section, gets larger proportional to ε^{-2} . As we showed in Section 2, Problem 2 is $\#P$ -hard even when i is small as fractional power of N . For small i , we have to set ε in Algorithm 1 very small as $\varepsilon \leq i/N$, and it makes Algorithm 1 inefficient. In this section, we propose another approximation algorithm for the i -th LI, especially for a small i . The algorithm approximately computes $g(p)$ for each $p \in P$. Then, we use the following oracle.

Oracle 2. (*RAS for COUNTING IDEALS.*) Given an arbitrary ε ($0 < \varepsilon < 1$), δ ($0 < \delta < 1$), and a poset P , Oracle returns $Z \in \mathbb{Z}_+$ which approximates $|\mathcal{D}(P)|$ satisfying

$$\Pr \left[\frac{|Z - |\mathcal{D}(P)||}{|\mathcal{D}(P)|} \leq \varepsilon \right] \geq 1 - \delta.$$

Let γ_2 denote the time required for Oracle 2. Oracle 2 is obtained from Oracle 1 in $\text{Poly}(\varepsilon^{-1}, -\ln \delta, |P|, \gamma_1)$ time, more precisely $O(\gamma_1 |P|^3 \varepsilon^{-2} \ln(|P|/\delta))$ with using a *self-reducibility*. See [10] for detail, and see also e.g. [9] about a relationship between sampling and approximate counting.

An essential idea of an approximation algorithm for the i -th LI is to compute an estimator $\hat{g}(p)$ for $g(p)$ for every $p \in P$, and to find a set $S \subseteq P$ satisfying $\hat{g}(p) < k$. Unfortunately, this simple idea cannot find an ideal $S \in \mathcal{D}(P)$, since an event of $\hat{g}(p) < k \leq \hat{g}(q)$ can happen to a pair $p \prec q$ with a non-negligible probability. The following algorithm gets rid of this issue.

Algorithm 2. (ε -estimator for the $f(N)$ -th LI.)

- 1 **Input:** A poset P , ε ($0 < \varepsilon < \lambda$), δ ($0 < \delta < 1$),
a uniformly contraction map $f : \mathbb{Z}_{++} \rightarrow \mathbb{Z}_{++}$.
- 2 **Compute** $\hat{N}$ approximating $|\mathcal{D}(p)|$ by Oracle 2.
(where $\hat{N}$ satisfies $\Pr[|\hat{N} - |\mathcal{D}(p)|| \leq (\varepsilon/3) \cdot |\mathcal{D}(p)|] \geq 1 - \delta/(2|P|)$).
- 3 **Set** $k := f(\hat{N})$
(thus k satisfies $\Pr[|k - |f(N)|| \leq (\varepsilon/3) \cdot |f(N)|] \geq 1 - \delta/(2|P|)$).
- 4 **Set** $S := \emptyset$, and $T := P$.
- 5 **While** $(\exists p \in T, \text{ s.t. } \forall q \in P, \text{ if } q \prec p \text{ then } q \in S)$ {
- 6 **Compute** $\hat{g}(p)$ approximating $g(p)$ by Oracle 2
(where $\hat{g}(p)$ satisfies $\Pr[|\hat{g}(p) - g(p)| \leq (\varepsilon/3) \cdot g(p)] \geq 1 - \delta/(2|P|)$).
- 7 **If** $\hat{g}(p) < k$ **then** $S := S \cup \{p\}$.
- 8 **Set** $T := T \setminus \{p\}$.
- 9 }
- 10 **Output** S and **halt**.

Theorem 3. Algorithm 2 outputs an ideal $S \in \mathcal{D}(P)$ in $O(|P| \cdot \gamma_2)$ time, and S satisfies

$$\Pr [S_{\lfloor (1-\varepsilon)f(N) \rfloor} \subseteq S \subseteq S_{\lceil (1+\varepsilon)f(N) \rceil}] \geq 1 - \delta.$$

Sketch of proof. The time complexity is easy to see. It is also easy to see that an output of Algorithm 2 is an ideal of P , since $p \in S$ implies that Algorithm 2 computed $\hat{g}(p)$, that is only when $q \in P$ ($\forall q \prec p$). We show that S satisfies the inequality (6). From the condition of Algorithm 2, k satisfies

$$\Pr \left[\frac{|k - f(N)|}{f(N)} > \frac{\varepsilon}{3} \right] < \frac{\delta}{2|P|}.$$

Then we obtain the following.

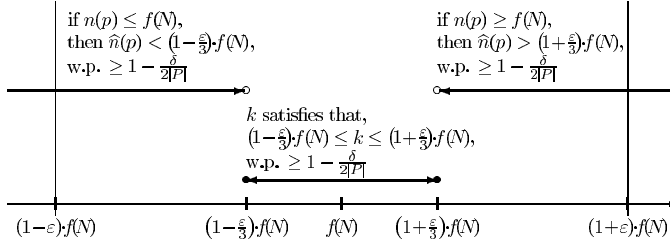


Fig. 3. A figure of the relationship between k and $\hat{g}(p)$

Claim 1. The probability that $k > (1 + \varepsilon/3) \cdot f(N)$ or $k < (1 - \varepsilon/3) \cdot f(N)$ is less than $\delta/(2|P|)$.

Suppose we have values $\hat{g}(p)$ for all $p \in P$, satisfying

$$\Pr \left[\frac{|\hat{g}(p) - g(p)|}{g(p)} \leq \frac{\varepsilon}{3} \right] \geq 1 - \frac{\delta}{2|P|}, \quad (6)$$

and $\hat{g}(p)$ coincident to the values in Algorithm 2 if it is computed.

Claim 2. For any $p \in P$,

Case 1. if $g(p) \geq (1 + \varepsilon) \cdot f(N)$, then the probability $\hat{g}(p) \leq (1 + (1/3)\varepsilon) \cdot f(N)$ is less than $\delta/(2|P|)$, and

Case 2. if $g(p) \leq (1 - \varepsilon) \cdot f(N)$, then the probability $\hat{g}(p) \geq (1 - (1/3)\varepsilon) \cdot f(N)$ is less than $\delta/(2|P|)$.

From Claim 1 and 2, we obtain the following (see Figure 3);

Claim 3. For any $p \in P$,

Case 1. if $g(p) \geq (1 + \varepsilon) \cdot f(N)$, then $\hat{g}(p) < k$ with a probability less than $\delta/|P|$, and

Case 2. if $g(p) \leq (1 - \varepsilon) \cdot f(N)$, then $\hat{g}(p) \geq k$ with a probability less than $\delta/|P|$.

Now we conclude the proof by showing (6). Algorithm 2 implies that if $S_{(1-\varepsilon) \cdot f(N)} \not\subseteq S$, then there exists a $p \in P$ satisfying $g(p) < (1 - \varepsilon) \cdot f(N)$, and exists a $q \in P$ satisfying $q \prec p$ and $\hat{g}(q) < k$. Note that $q \prec p$ means $g(q) < g(p)$, thus the above claim can be simply transformed into that if $S_{(1-\varepsilon) \cdot f(N)} \not\subseteq S$, then there exists a $q \in P$ satisfying $g(q) < (1 - \varepsilon) \cdot f(N)$ and $\hat{g}(q) < k$. From Case 1 of Claim 3, the probability of $S_{(1-\varepsilon) \cdot f(N)} \not\subseteq S$ satisfies that

$$\begin{aligned} \Pr [S_{(1-\varepsilon) \cdot f(N)} \not\subseteq S] &\leq \sum_{p \in S_{(1-\varepsilon) \cdot f(N)}} \Pr [p \notin S] \\ &\leq |\{p \mid g(p) < (1 - \varepsilon) \cdot f(N)\}| \cdot \frac{\delta}{|P|}. \end{aligned}$$

In a similar way, we can show

$$\Pr [S \not\subseteq S_{(1+\varepsilon) \cdot f(N)}] \leq |\{p \mid g(p) \geq (1 + \varepsilon) \cdot f(N)\}| \cdot \frac{\delta}{|P|}.$$

Since the sets $\{p \mid g(p) < (1 - \varepsilon) \cdot f(N)\}$ and $\{p \mid g(p) \geq (1 + \varepsilon) \cdot f(N)\}$ are disjoint, we obtain

$$\Pr [S_{(1-\varepsilon)f(N)} \subseteq Z \subseteq S_{(1+\varepsilon)f(N)}] \geq 1 - |P| \cdot \frac{\delta}{|P|} = 1 - \delta. \quad \blacksquare$$

6 Concluding Remarks

We proposed randomized approximation schemes for the i -th LI. When a poset is in some special classes such as series-parallel, bounded width, etc., we can find the i -th LI exactly in polynomial time by Steiner's result [22]. We can also show that the existence of a fully polynomial time randomized approximation scheme for finding the i -th LI iff the existence of FPRAS for counting ideals of a poset (see [10] for detail). The existence of a polynomial time sampler on ideals (or antichains) of a poset, or an FPRAS for counting, is open. It is also open whether or not Problems 1 and 2 are in NP.

Acknowledgment

The authors thank Shin-Ichi Nakano and Christine Cheng for their helpful comments. The first author is supported by Grant-in-Aid for Scientific Research.

References

1. Bhatnagar, N., Greenberg, S., Randall, D.: Sampling stable marriages: why the spouse-swapping won't work. In: Proc. of SODA 2008, pp. 1223–1232 (2008)
2. Blair, C.: Every finite distributive lattice is a set of stable matchings. J. Comb. Theory A 37, 353–356 (1984)
3. Cheng, C.T.: The generalized median stable matchings: finding them is not that easy. In: Proc. of Latin 2008, pp. 568–579 (2008)
4. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (2002)
5. Dubhashi, D.P., Mehlhorn, K., Rajan, D., Thiel, C.: Searching, sorting and randomised algorithms for central elements and ideal counting in posets. In: Shyammasundar, R.K. (ed.) FSTTCS 1993. LNCS, vol. 761, pp. 436–443. Springer, Heidelberg (1993)
6. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Month. 69, 9–15 (1962)
7. Gusfield, D., Irving, R.W.: The Stable Marriage Problem, Structure and Algorithms. MIT Press, Cambridge (1989)
8. Irving, R.W., Leather, P.: The complexity of counting stable marriages. SIAM J. Comput. 15, 655–667 (1986)
9. Jerrum, M.: Counting, Sampling and Integrating: Algorithms and Complexity. ETH Zürich, Birkhauser, Basel (2003)
10. Kijima, S., Nemoto, T.: Randomized approximation for generalized median stable matching. RIMS-preprint 1648 (2008)

11. Klaus, B., Klijn, F.: Median stable matching for college admission. *Int. J. Game Theory* 34, 1–11 (2006)
12. Klaus, B., Klijn, F.: Smith and Rawls share a room: stability and medians. Meteor RM/08-009, Maastricht University (2008), <http://edocs.uu.unimaas.nl/loader/file.asp?id=1307>
13. Knuth, D.: *Stable Marriage and Its Relation to Other Combinatorial Problems*. American Mathematical Society (1991)
14. Nemoto, T.: Some remarks on the median stable marriage problem. In: *ISMP 2000* (2000)
15. Propp, J., Wilson, D.B.: Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Struct. Algo.* 9, 223–252 (1996)
16. Provan, J.S., Ball, M.O.: The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.* 12, 777–788 (1983)
17. Roth, A.E., Sotomayor, M.A.O.: *A Two-Sided Matchings: A Study In Game-Theoretic Modeling And Analysis*. Cambridge University Press, Cambridge (1990)
18. Schwarz, M., Yenmez, M.B.: Median stable matching. NBER Working Paper No. w14689 (2009)
19. Sethuraman, J., Teo, C.P., Qian, L.: Many-to one stable matching: geometry and fairness. *Math. Oper. Res.* 31, 581–596 (2006)
20. Squire, M.B.: Enumerating the ideals of a poset. preprint, North Carolina State University (1995)
21. Steiner, G.: An algorithm for generating the ideals of a partial order. *Oper. Res. Lett.* 5, 317–320 (1986)
22. Steiner, G.: On the complexity of dynamic programming for sequencing problems with precedence constraints. *Ann. Oper. Res.* 26, 103–123 (1990)
23. Teo, C.P., Sethuraman, J.: The geometry of fractional stable matchings and its applications. *Math. Oper. Res.* 23, 874–891 (1998)

A Polynomial-Time Perfect Sampler for the Q -Ising with a Vertex-Independent Noise

M. Yamamoto¹, S. Kijima², and Y. Matsui³

¹ Dept. of Mathematical Sciences, School of Science, Tokai University
yamamoto@tokai-u.jp

² Research Institute for Mathematical Sciences, Kyoto University
kijima@kurims.kyoto-u.ac.jp

³ Dept. of Mathematical Sciences, School of Science, Tokai University
yasuko@ss.u-tokai.ac.jp

Abstract. We present a polynomial-time perfect sampler for the Q -Ising with a vertex-independent noise. The Q -Ising, one of the generalized models of the Ising, arose in the context of Bayesian image restoration in statistical mechanics. We study the distribution of Q -Ising on a two-dimensional square lattice over n vertices, that is, we deal with a discrete state space $\{1, \dots, Q\}^n$ for a positive integer Q . Employing the Q -Ising (having a parameter β) as a prior distribution, and assuming a Gaussian noise (having another parameter α), a posterior is obtained from the Bayes' formula. Furthermore, we generalize it: the distribution of noise is not necessarily a Gaussian, but any vertex-independent noise. We first present a Gibbs sampler from our posterior, and also present a perfect sampler by defining a coupling via a monotone update function. Then, we show $O(n \log n)$ mixing time of the Gibbs sampler for the generalized model under a condition that β is sufficiently small (whatever the distribution of noise is). In case of a Gaussian, we obtain another more natural condition for rapid mixing that α is sufficiently larger than β . Thereby, we show that the expected running time of our sampler is $O(n \log n)$.

1 Introduction

The Markov chain Monte Carlo (MCMC) method is a popular tool for sampling from a desired probability distribution. The probability distribution is defined by constructing a (an ergodic) Markov chain so that its (unique) stationary distribution is the desired probability distribution. We then run the chain repeatedly, that is, start at an arbitrary initial state, and repeatedly change the current state according to the transition probabilities. The state after a large number of iterations is used as a sample from the probability distribution. The Gibbs sampler, which is used in this paper, is one of the well-known MCMC algorithms.

The sample generated by this simple method is just an approximation: the precision of approximation is often measured by *total variation distance*. The *mixing time* of a sampling algorithm is the number t such that how many iterations t are needed to converge to the target stationary distribution within

a prescribed (or an acceptable) precision. The main drawback of this simple method is in a practical issue: practitioners implementing this algorithm have to know the mixing time. For getting around this problem, Propp and Wilson [6] proposed a sampling algorithm which does not take any information about the convergence rate beforehand. This was achieved by *coupling from the past*, where how many (coupling) steps we need is automatically determined. Moreover, this algorithm produces an exact sampling from the target distribution. That's why this algorithm was called an *exact sampling*, which is now called a *perfect sampling*.

In this paper, we present a polynomial-time perfect sampler for the Q -Ising with a vertex-independent noise. The Q -Ising is one of the generalized models of the Ising. (The Q -Ising for $Q = 2$ is the Ising.) We study the Q -Ising on the two-dimensional square lattice. Throughout this paper, we denote by n the number of vertices of a square lattice. In the Q -Ising, vertices of a square lattice take on discrete Q values, say, $\{1, \dots, Q\}$, while vertices in the *Ising* take on binary values, say, $\{-1, +1\}$.

The motivation of the Q -Ising comes from Bayesian image restoration studied in statistical mechanics: the original image that has n pixels, each of which has Q grey-scales, is assumed to be generated from the Q -Ising over n vertices. Initially, Geman and Geman [2] proposed a Gibbs sampler for Bayesian restoration of a black-and-white (i.e., two valued) image, adopting the Ising as a prior distribution. Inoue and Carlucci [5] investigated static and dynamic properties of gray-scale image restoration by making use of the Q -Ising. They checked the efficiency of the model by Monte Carlo simulations as well as an iterative algorithm using mean-field approximation. Tanaka et al. [7] proposed an algorithm based on Bethe approximation to estimate hyperparameters (that are used for image restoration) when the Q -Ising is adopted as a prior distribution.

In [3], Gibbs showed a perfect sampler for the Ising with a Gaussian noise. Given a square lattice $G = (V, E)$ over n vertices, the prior distribution is assumed to follow the Ising: any $x \in \{-1, +1\}^n$ is generated with $\Pr\{X = x\} = e^{-\beta H(x)} / Z_\beta$ for some $\beta > 0$, where $H(x) = -\sum_{(i,j) \in E} x_i x_j$, and Z_β is a normalizing constant. The value of β reflects the strength of the attractive force between adjacent vertices. The distribution of noise at each vertex is assumed to independently follow a normal (or Gaussian) distribution $N(0, \sigma^2)$ of mean zero and variance σ^2 . From the Bayes' formula, the posterior of x given y is defined as follows:

$$\Pr\{X = x | Y = y\} = \frac{1}{Z_{\sigma, \beta}(y)} \exp \left(\frac{1}{2\sigma^2} \sum_{i \in V} x_i y_i + \beta \sum_{(i,j) \in E} x_i x_j \right), \quad (1)$$

where $Z_{\sigma, \beta}(y)$ is a normalizing constant. Then, it was shown that the mixing time of a Gibbs sampler from (1) is $O(n^2)$, which was improved to $O(n \log n)$ in [4, section 4]. Moreover, Gibbs showed a *monotone* coupling, thereby derived a perfect sampler that has the expected running time $O(n \log n)$.

Remark 1. Here, it is necessary to give some comments on [4], in particular, section 3 of the paper. Gibbs obtained $O(n \log n)$ mixing time for a *continuous* state space, say, $[0, 1]^n$, while we deal with a *discrete* state space. It seems non-trivial whether the argument in [4] can be extended to the discrete state space, say, $\{1, \dots, Q\}^n$ for *any* fixed positive integer Q . (The similar analysis might be applied to $\{1, \dots, Q\}^n$ for a sufficiently large Q .) With the practical motivation in mind, it is natural to study a distribution over a discrete state space.

In this paper, we employ the Q -Ising as a prior distribution to deal with a discrete state space. In the similar way to obtaining (1), we can derive a posterior, that has two parameters: α related to a Gaussian noise and β related to the Q -Ising. (This posterior is also appeared explicitly in [5].) Furthermore, we generalize it: the distribution of noise is not necessarily a Gaussian, but *any* vertex-independent distribution. See the next section for the details. We first present a Gibbs sampler from our posterior, and also present a perfect sampler by defining a coupling via an *update function*. We then show that it is monotone. Finally, we show $O(n \log n)$ mixing time of the Gibbs sampler for the generalized model under a condition that β is sufficiently small. In case of a Gaussian, we obtain another more natural condition that α is sufficiently larger than β . Thereby, we derive the following our main theorems:

Theorem 1 (vertex-independent noise). *Let $\mathcal{D}_1$ be a posterior of the Q -Ising with an arbitrary vertex-independent noise D . For any positive integer Q and for any distribution D , we have the following: if $\beta > 0$ satisfies*

$$\beta \leq \frac{\ln(8Q) - \ln(8Q - 1)}{2Q} \quad (\beta = O(1/Q^2)),$$

then there exists a perfect sampler for $\mathcal{D}_1$ that has the expected running time $O(n \log n)$.

Theorem 2 (Gaussian noise). *Let $\mathcal{D}_2$ be a posterior of the Q -Ising with a Gaussian noise. For any positive integer Q , and for any $\alpha, \beta > 0$ satisfying*

$$\alpha \geq 8Q^2\beta + 3 \ln(Q/2) \quad (\alpha = \Omega(Q^2)\beta + \Omega(\ln Q)),$$

there exists a perfect sampler for $\mathcal{D}_2$ that has the expected running time $O(n \log n)$.

Remark 2. The former theorem says that if β is sufficiently small, e.g., $\beta = O(1/Q^2)$, then a polynomial-time perfect sampler exists *whatever the distribution D is*. On the other hand, the latter says in case that D is a Gaussian, if α is suitably larger than β , then a polynomial-time perfect sampler exists even if $\beta = \Omega(1/Q^2)$. Gibbs showed (for the continuous version) that if $\alpha \geq (3/4)\beta$, then a polynomial-time perfect sampler exists.

2 The Probability Model and the Markov Chain

2.1 The Probability Model

As is stated in the introduction, we consider the Q -Ising as a prior distribution, which is defined as follows: Given any two-dimensional square lattice $G = (V, E)$,

let $\Xi = \{1, \dots, Q\}^V$. (From now on, we denote $\{1, \dots, Q\}$ by $[Q]$.) Then, for any $x \in \Xi$, the distribution is defined as

$$\Pr\{X = x\} \stackrel{\text{def.}}{=} \frac{\exp(-H_\beta(x))}{Z_\beta}, \quad \text{where} \quad \begin{cases} H_\beta(x) = \beta \sum_{(u,v) \in E} (x(u) - x(v))^2, \\ Z_\beta = \sum_{x \in \Xi} \exp(-H_\beta(x)), \end{cases}$$

where $x(v) \in [Q]$ is the value of $x \in \Xi$ at $v \in V$. We assume that the distribution of the noise at each vertex independently follows a common distribution, here denoted by D . That is, for a given $X = x \in \Xi$, the distribution of the output $Y = y \in \Xi$ caused by this degradation process is

$$\begin{aligned} \Pr\{Y = y|X = x\} &= \prod_{v \in V} \Pr\{Y(v) = y(v)|X(v) = x(v)\} \\ &= \exp\left(\sum_{v \in V} \ln D(x(v), y(v))\right). \end{aligned}$$

In case D is a normal (or Gaussian) distribution $N(0, \sigma^2)$ of mean zero and variance σ^2 , then

$$\Pr\{Y = y|X = x\} = \frac{1}{Z_\sigma} \exp\left(-\frac{\sum_{v \in V} (x(v) - y(v))^2}{2\sigma^2}\right),$$

where Z_σ is a normalizing constant. Then, the posterior is obtained from the two distributions defined above using the Bayes' formula:

$$\Pr\{X = x|Y = y\} = \frac{\Pr\{Y = y|X = x\} \Pr\{X = x\}}{\Pr\{Y = y\}}.$$

Fix $y \in \Xi$ arbitrarily. Then, the denominator of the Bayes' formula is a constant. The numerator is

$$\begin{aligned} &\Pr\{Y = y|X = x\} \Pr\{X = x\} \\ &= \exp\left(\sum_{v \in V} \ln D(x(v), y(v))\right) \cdot \frac{1}{Z_\beta} \exp\left(-\beta \sum_{(u,v) \in E} (x(u) - x(v))^2\right) \\ &= \frac{1}{Z_\beta} \exp\left(\sum_{v \in V} \ln D(x(v), y(v)) - \beta \sum_{(u,v) \in E} (x(u) - x(v))^2\right). \end{aligned}$$

Thus, the posterior which we study in this paper is given by

$$\Pr\{X = x|Y = y\} = \frac{1}{Z_{D,\beta}(y)} \cdot \exp(-H_{D,\beta}(x, y)),$$

where

$$H_{D,\beta}(x, y) = -\sum_{v \in V} \ln D(x(v), y(v)) + \beta \sum_{(u,v) \in E} (x(u) - x(v))^2,$$

and $Z_{D,\beta}(y)$ is a normalizing constant so that $\sum_{x \in \Xi} \Pr\{X = x | Y = y\} = 1$. In case D is $N(0, \sigma^2)$, the posterior is given by

$$H_{D,\beta}(x, y) = H_{\alpha,\beta}(x, y) = \alpha \sum_{v \in V} (x(v) - y(v))^2 + \beta \sum_{(u,v) \in E} (x(u) - x(v))^2,$$

where $\alpha = 1/(2\sigma^2)$, and we denote the normalizing constant by $Z_{\alpha,\beta}(y)$.

2.2 The Markov Chain

In what follows, we fix $y \in \Xi$ arbitrarily. We define a Markov chain by presenting a Gibbs sampler (for the fixed y) from the posterior defined above. Let $\mathcal{M}$ be the Markov chain. The state space of $\mathcal{M}$ is Ξ . Then, the transition probabilities are defined by the Gibbs sampler shown in Fig. 1 below, where $x^{(i)}$ indicates $x^{(i)}(w) = x(w)$ for all $w \in V \setminus \{v\}$ and $x^{(i)}(v) = i$.

step 0 : Given $x \in \Xi$,

step 1 : Choose $v \in V$ uniformly.

step 2 : Set $x'(w) = x(w)$ for all $w \in V \setminus \{v\}$, and let for each $j \in [Q]$,

$$x'(v) = j \text{ with probability } \frac{p_j}{\sum_{i \in [k]} p_i}, \quad \text{where } p_i \stackrel{\text{def.}}{=} \frac{\exp\left(-H_{D,\beta}\left(x^{(i)}, y\right)\right)}{Z_{D,\beta}(y)}.$$

Fig. 1. The Gibbs sampler from our posterior

It is easy to see that $\mathcal{M}$ is a finite ergodic Markov chain, and hence it has a unique stationary distribution. Moreover, the stationary distribution exactly follows our posterior. (This is a well-known property of the Gibbs sampler.) Let v be a vertex chosen at step 1 of the Gibbs sampler. Then, since for any $i, i' \in [Q]$ we have $x^{(i)}(w) = x^{(i')}(w)$ for any $w \in V \setminus \{v\}$, we have the following from an elementary calculation: for any $j \in [Q]$,

$$\frac{p_j}{\sum_{i \in [Q]} p_i} = \frac{\exp\left(\ln D(j, f(v)) - \beta \sum_{w \in N(v)} (j - x^{(j)}(w))^2\right)}{\sum_{i \in [Q]} \exp\left(\ln D(i, f(v)) - \beta \sum_{w \in N(v)} (i - x^{(i)}(w))^2\right)},$$

where $N(v)$ is the set of vertices adjacency to v . In case D is $N(0, \sigma^2)$,

$$\frac{p_j}{\sum_{i \in [Q]} p_i} = \frac{\exp\left(-(\alpha(j - f(v))^2 + \beta \sum_{w \in N(v)} (j - x^{(j)}(w))^2)\right)}{\sum_{i \in [Q]} \exp\left(-(\alpha(i - f(v))^2 + \beta \sum_{w \in N(v)} (i - x^{(i)}(w))^2)\right)}.$$

Here, we define a cumulative distribution function $q_v^{(x)}(j)$ of $p_j / \sum_{i \in [Q]} p_i$ for later use: $q_v^{(x)}(0) = 0$ and for any $j \in [Q]$, $q_v^{(x)}(j) \stackrel{\text{def.}}{=} \sum_{i' \in [j]} p_{i'} / (\sum_{i \in [Q]} p_i)$.

3 The Perfect Sampler

3.1 The Monotone Coupling from the Past

Before presenting our sampling algorithm, we briefly review the *coupling from the past* (abbrev. *CFTP*) proposed in [6], in particular, the *monotone CFTP*.

Given an ergodic Markov chain with a finite state space Ξ and a transition matrix P . The transition probabilities can be described by defining a deterministic function $\phi : \Xi \times [0, 1) \rightarrow \Xi$ as well as a random number λ uniformly distributed over $[0, 1)$ so that $\Pr(\phi(x, \lambda) = y) = P(x, y)$ for every pair of $x, y \in \Xi$. This function is called an *update function*. Then, we can realize the Markov chain $X \mapsto X'$ by setting $X' = \phi(X, \lambda)$. Note that an update function corresponding to the given transition matrix P is not unique. For integers t_1 and t_2 ($t_1 < t_2$), let $\lambda = (\lambda[t_1], \lambda[t_1 + 1], \dots, \lambda[t_2 - 1]) \in [0, 1)^{t_2 - t_1}$ be a sequence of random real numbers. Given an initial state x , the result of transitions of the chain from time t_1 to time t_2 by ϕ with λ is denoted by $\Phi_{t_1}^{t_2}(x, \lambda) : \Xi \times [0, 1)^{t_2 - t_1} \rightarrow \Xi$, where $\Phi_{t_1}^{t_2}(x, \lambda) \stackrel{\text{def.}}{=} \phi(\phi(\dots(\phi(x, \lambda[t_1]), \dots), \lambda[t_2 - 2]), \lambda[t_2 - 1])$.

Suppose that there exists a partial order “ $\succeq$ ” on the state space Ξ . We say that an update function ϕ is *monotone with respect to $\succeq$* if $\forall \lambda \in [0, 1), \forall x, y \in \Xi$ $[x \succeq y \implies \phi(x, \lambda) \succeq \phi(y, \lambda)]$. We also say that a Markov chain is *monotone* if the chain has a monotone update function. Suppose further that there exist a unique maximum state $x_{\max}$ and a unique minimum state $x_{\min}$ with respect to $\succeq$, that is, there exists a pair of $x_{\max}$ and $x_{\min}$ such that $x_{\max} \succeq x \succeq x_{\min}$ for all $x \in \Xi \setminus \{x_{\max}, x_{\min}\}$. Then, a standard monotone coupling from the past (CFTP) algorithm is expressed as in Fig. 2. Then, the monotone CFTP theorem says:

- step 1:** Set the starting time period as $T = -1$, and set λ as the empty sequence.
- step 2:** Generate random real numbers $\lambda[T], \lambda[T + 1], \dots, \lambda[\lceil T/2 \rceil - 1]$ uniformly from $[0, 1)$, and insert them to the head of λ in order, i.e., set λ as $\lambda = (\lambda[T], \lambda[T + 1], \dots, \lambda[-1])$.
- step 3:** Start two chains from $x_{\max}$ and $x_{\min}$ respectively at time period T , and run each chain to time period 0 by the update function ϕ with λ . (Here we note that each chain uses the common sequence λ .)
- step 4:** For two states $\Phi_T^0(x_{\max}, \lambda)$ and $\Phi_T^0(x_{\min}, \lambda)$,
 - (a) If $\exists y \in \Xi$ $[y = \Phi_T^0(x_{\max}, \lambda) = \Phi_T^0(x_{\min}, \lambda)]$, then return y .
 - (b) Else, set the starting time period T as $T = 2T$, and go to **step 2**.

Fig. 2. The monotone CFTP algorithm

Theorem 3 (Monotone CFTP Theorem [6]). *Given a monotone Markov chain as above. The monotone CFTP algorithm shown in Fig. 2 terminates with probability 1, Moreover, the output exactly follows the stationary distribution of the Markov chain.*

With these preparation above, we now describe our sampling algorithm. For this, it suffices to define an update function ϕ for our posterior. Besides a random number $\lambda \in [0, 1)$, our update function $\phi : \Xi \times V \times [0, 1) \rightarrow \Xi$ takes $v \in V$ chosen uniformly from V . Then, given $x \in \Xi$, the new state $x' = \phi(x, v, \lambda)$ is defined as follows: recall our cumulative distribution function $q_v^{(x)}(j)$ defined in the previous section. Let $i \in [Q]$ be an integer satisfying $q_v^{(x)}(i-1) \leq \lambda < q_v^{(x)}(i)$. Then, for each $w \in V$, set $x'(w) = i$ if $w = v$, and $x'(w) = x(w)$ otherwise.

3.2 The Monotone Markov Chain

For showing the monotonicity of our update function, we introduce a natural partial order “ $\succeq$ ” to Ξ . For an arbitrary pair of $x, y \in \Xi$, we say that $x \succeq y$ if $x(w) \geq y(w)$ for all $w \in V$. Let $x_{\max}$ (resp. $x_{\min}$) be a state such that $x_{\max}(w) = Q$ (resp. $x_{\min}(w) = 1$) for all w . Then, $x_{\max}$ (resp. $x_{\min}$) is the unique maximum (resp. minimum) of the partially ordered set Ξ w.r.t. $\succeq$.

Lemma 1. *Let $x, y \in \Xi$ be arbitrary states such that $x \succeq y$. Let $v \in V$ be an arbitrary vertex. Then, for any $\alpha, \beta > 0$, and for any $j \in [Q]$, we have $q_v^{(x)}(j) < q_v^{(y)}(j)$.*

Proof. Fix $j \in [Q]$ arbitrarily. By some elementary calculation, we have $q_v^{(x)}(j) < q_v^{(y)}(j)$ if for any $s, t : 1 \leq s \leq j < t \leq Q$,

$$\begin{aligned} & \exp \left(- \left(\beta \sum_{w \in N(v)} (s - x^{(s)}(w))^2 \right) \right) \cdot \exp \left(- \left(\beta \sum_{w \in N(v)} (t - y^{(t)}(w))^2 \right) \right) \\ & < \exp \left(- \left(\beta \sum_{w \in N(v)} (s - y^{(s)}(w))^2 \right) \right) \cdot \exp \left(- \left(\beta \sum_{w \in N(v)} (t - x^{(t)}(w))^2 \right) \right). \end{aligned}$$

Furthermore, for any such fixed s, t , this inequality holds if for any $w \in N(v)$,

$$(s - x^{(s)}(w))^2 + (t - y^{(t)}(w))^2 > (s - y^{(s)}(w))^2 + (t - x^{(t)}(w))^2.$$

Since $x^{(i)}(w) \geq y^{(i)}(w)$ for any $i \in [Q]$, this inequality holds if $t > s$, which is the assumption on s and t . $\square$

Theorem 4. *Our update function ϕ is monotone on the partially ordered set Ξ w.r.t. $\succeq$, i.e., $\forall x, y \in \Xi, \forall v \in V, \forall \lambda \in [0, 1) [x \succeq y \implies \phi(x, v, \lambda) \succeq \phi(y, v, \lambda)]$.*

Proof. Let $x, y \in \Xi$ be arbitrary states such that $x \succeq y$. Fix $v \in V$ and $\lambda \in [0, 1)$ arbitrarily. First, it is easy to see from the definition of ϕ that $x'(w) \geq y'(w)$ for every $w \in V \setminus \{v\}$. Next, from the above lemma, $q_v^{(x)}(j) < q_v^{(y)}(j)$ for any $j \in [Q]$. From this and the definition of ϕ , we also have $x'(v) \geq y'(v)$. Therefore, $x'(w) \geq y'(w)$ for every $w \in V$, and hence we conclude that $\phi(x, v, \lambda) \succeq \phi(y, v, \lambda)$. $\square$

4 Expected Running Time

Before showing the expected running time of our sampling algorithm, we note notions and notations. For probability distribution p_1 and p_2 , the *total variation distance* between p_1 and p_2 is defined as $d_{\text{TV}}(p_1, p_2) \stackrel{\text{def.}}{=} (1/2) \sum_{x \in \Xi} |p_1(x) - p_2(x)|$. Consider an ergodic Markov chain over a finite state space Ξ . Given a precision $\epsilon > 0$, the *mixing time* $\tau(\epsilon)$ of the Markov chain is defined as $\tau(\epsilon) \stackrel{\text{def.}}{=} \max_{x \in \Xi} \{ \min\{t : \forall s \geq t [d_{\text{TV}}(\pi, P_x^s) \leq \epsilon]\} \}$, where π is the stationary distribution, and P_x^s is the probability distribution of the chain at time s where the chain starts at x . The *path coupling lemma* [1] is a powerful tool for bounding the mixing time.

Theorem 5 (Path coupling lemma [1]). *Let Z_t be an ergodic Markov chain on a finite state space Ξ . Let $d : \Xi \times \Xi \rightarrow \{0, 1, \dots, D\}$ be a (quasi-)metric function for some integer D . Let $S \subset \Xi \times \Xi$ be a set such that graph (Ξ, S) is connected. Suppose that there exists a (partial) coupling (X_t, Y_t) for Z_t such that*

$$\gamma < 1, \forall (z, z') \in S [E[d(X_1, Y_1) | X_0, Y_0] \leq \gamma E[d(X_0, Y_0)]] .$$

Then, $\tau(\epsilon) \leq \ln(D/\epsilon)/(1 - \gamma)$.

In this section, we estimate the expected running time of our sampling algorithm. For this, we first estimate the mixing time of the Gibbs sampler shown in Fig. 1 by the path coupling lemma above, where the coupling is the one implicitly specified in our sampling algorithm shown in Fig. 2.

4.1 Vertex-Independent Noise

In this subsection, we show the mixing time, and derive a condition for rapid mixing in case the distribution of noise is any vertex-independent noise.

Lemma 2. *For any positive integer Q and for any distribution D , if $\beta > 0$ satisfies $\beta \leq (\ln(8Q) - \ln(8Q - 1))/(2Q)$, then the mixing time $\tau(\epsilon)$ of the Gibbs sampler shown in Fig. 1 is bounded by $\tau(\epsilon) \leq 2n \ln(Qn/\epsilon)$.*

Proof. As stated above, we prove it by the path coupling lemma, where the coupling is the one implicitly specified in our sampling algorithm shown in Fig. 2. We will show that $E[d(X_1, Y_1) | X_0 = x_0, Y_0 = y_0] \leq 1 - 1/(2n)$ for any $x_0, y_0 \in \Xi$ with $d(x_0, y_0) = 1$, where $d(x, y) \stackrel{\text{def.}}{=} \sum_{v \in V} |x(v) - y(v)|$. We assume that X_0 and Y_0 do not agree at $v_0 \in V$. We denote by v the vertex chosen at step 1 in the Gibbs sampler shown in Fig. 1.

First, consider the case of $v = v_0$. This event occurs with probability $1/n$. In this case, the coupling is identical since $x(w) = y(w)$ for all $w \in V \setminus \{v\}$. Moreover, the distance decreases by one, i.e, it gets zero.

Next, consider the case of $v \neq v_0$. In this case, the coupling is identical. However, in contrast to the first case, the distance does not change, i.e., it remains one.

Finally, consider the case of $v \in N(v_0)$. This event occurs with probability at most $4/n$. Recall the coupling by the update function: Given $X = x$ and $Y = y$, choose λ uniformly from $[0, 1)$. Then, we define $x'(v)$ and $y'(v)$ as

$$\begin{aligned} x'(v) &= \ell \text{ where } q_v^{(x)}(\ell - 1) \leq \lambda < q_v^{(x)}(\ell), \\ y'(v) &= \ell \text{ where } q_v^{(y)}(\ell - 1) \leq \lambda < q_v^{(y)}(\ell). \end{aligned}$$

In what follows, we assume w.l.o.g. that $X_0(v_0) = Y_0(v_0) + 1$. We will need the following propositions:

Proposition 1

$$E[d(X_1, Y_1) - 1 | X_0, Y_0, v \in N(v_0)] = \sum_{j \in [Q]} \left(q_v^{(y)}(j) - q_v^{(x)}(j) \right).$$

Proposition 2. *If $\beta > 0$ satisfies $\beta \leq (\ln(8Q) - \ln(8Q - 1))/(2Q)$, then $\sum_{j \in [Q]} (q_v^{(y)}(j) - q_v^{(x)}(j)) \leq 1/8$.*

Here, we omit the proofs of these two propositions. From these two, we have $E[d(X_1, Y_1) - 1 | X_0, Y_0, v \in N(v_0)] \leq 1/8$. Therefore, the total expectation is

$$E[d(X_1, Y_1) - 1 | X_0, Y_0] \leq \frac{1}{n}(-1) + \frac{4}{n} \cdot \frac{1}{8} = -\frac{1}{2n}.$$

Since the maximum distance is Qn , this lemma follows from the path coupling lemma. $\square$

Our first theorem, Theorem 1, is derived from this lemma.

4.2 Gaussian Noise

In this subsection, we derive another condition for rapid mixing in case the distribution of noise is a Gaussian noise $N(0, \sigma^2)$.

Lemma 3. *For any positive integer Q , and for any $\alpha, \beta > 0$ satisfying $\alpha \geq 8Q^2\beta + 3\ln(Q/2)$, the mixing time $\tau(\epsilon)$ of the Gibbs sampler shown in Fig. 1 is bounded by $\tau(\epsilon) \leq 2n \ln(Qn/\epsilon)$.*

Proof. The proof is identical to the one for the general noise, except for using the following proposition instead of Proposition 2.

Proposition 3. *If $\alpha, \beta > 0$ satisfy the following inequality: $\alpha \geq 8Q^2\beta + 3\ln(Q/2)$, then $\sum_{j \in [Q]} (q_v^{(y)}(j) - q_v^{(x)}(j)) \leq 1/8$.*

Here, we omit the proof of this proposition. From Proposition 1 in the proof for the general noise and the proposition above, we obtain

$$E[d(X_1, Y_1) - 1 | X_0, Y_0, v \in N(v_0)] \leq \frac{1}{8}.$$

From this, we obtain the desired mixing time. $\square$

Our second theorem, Theorem 2, is derived from this lemma.

5 Conclusion

We have presented a polynomial time perfect sampler for our posterior, that is, for the Q -Ising with a vertex-independent noise. We have shown $O(n \log n)$ mixing time of the Gibbs sampler for the generalized model under a certain condition, e.g., $\beta = O(1/Q^2)$. This holds whatever the distribution of noise is. In case of a Gaussian, we obtain another more natural condition, e.g., $\alpha = \Omega(Q^2)\beta + \Omega(\ln Q)$. The problem is that these conditions are somewhat restrictive, in particular, in case of a Gaussian, we need $\alpha = \Omega(\ln Q)$ for rapid mixing however small β (but $\beta = \Omega(1/Q^2)$) is. For the similar but *continuous* state space in [4], Gibbs derived a comparable condition, $\alpha \geq (3/4)\beta$, for example. It is one of our future work to figure out whether our result holds for a range of $\alpha \geq \Omega(1)\beta$.

Acknowledgement

We thank Prof. Osamu Watanabe for having a useful discussion on MCMC in the context of Bayesian image restoration. The second author is supported by Grant-in-Aid for Scientific Research.

References

1. Bubley, R., Dyer, M.: Path coupling: A technique for proving rapid mixing in Markov chains. In: Proc. of FOCS 1997, pp. 223–231 (1997)
2. Geman, S., Geman, D.: Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans. Pattern Analysis and Machine Intelligence 6, 721–741 (1984)
3. Gibbs, A.L.: Bounding the convergence time of the Gibbs sampler in Bayesian image restoration. Biometrika 87(4), 749–766 (2000)
4. Gibbs, A.L.: Convergence in the Wasserstein metric for Markov chain Monte Carlo algorithms with applications to image restoration. Stochastic Models 20, 473–492 (2004)
5. Inoue, J., Carlucci, D.M.: Image restoration using the Q -Ising spin glass. Phys. Rev. E 64, 036121 (2001)
6. Propp, J., Wilson, D.: Exact sampling with coupled Markov chains and applications to statistical mechanics. Random Struct. and Algo. 9, 223–252 (1996)
7. Tanaka, K., Inoue, J., Titterton, D.M.: Probabilistic image processing by means of the Bethe approximation for the Q -Ising model. J. Phys. A: Math. Gen. 36, 11023–11035 (2003)

Extracting Computational Entropy and Learning Noisy Linear Functions

Chia-Jung Lee¹, Chi-Jen Lu², and Shi-Chun Tsai¹

¹ Department of Computer Science, National Chiao-Tung University,
Hsinchu, Taiwan

{leecj,sctsaai}@csie.nctu.edu.tw

² Institute of Information Science, Academia Sinica, Taipei, Taiwan
cjlu@iis.sinica.edu.tw

Abstract. We study the task of deterministically extracting randomness from sources containing computational entropy. The sources we consider have the form of a conditional distribution $(f(\mathcal{X})|\mathcal{X})$, for some function f and some distribution $\mathcal{X}$, and we say that such a source has computational min-entropy k if any circuit of size 2^k can only predict $f(x)$ correctly with probability at most 2^{-k} given input x sampled from $\mathcal{X}$. We first show that it is impossible to have a seedless extractor to extract from one single source of this kind. Then we show that it becomes possible if we are allowed a seed which is weakly random (instead of perfectly random) but contains some statistical min-entropy, or even a seed which is not random at all but contains some computational min-entropy. This can be seen as a step toward extending the study of multi-source extractors from the traditional, statistical setting to a computational setting. We reduce the task of constructing such extractors to a problem in learning theory: learning linear functions under arbitrary distribution with adversarial noise. For this problem, we provide a learning algorithm, which may have interest of its own.

1 Introduction

Randomness has become a useful tool in computer science, as the most efficient algorithms known for many important problems are randomized. However, when analyzing the performance of a randomized algorithm, we usually assume that the algorithm has access to a perfectly random source. In reality, the random sources we have access to are usually not perfect but may contain some amount of randomness. The amount of randomness in a source is usually measured by its min-entropy, where a source has min-entropy k if every element occurs with probability at most 2^{-k} . From a source with some min-entropy, we would like to have a procedure, called an *extractor* [20,28], to extract almost perfect randomness, which can then be used for randomized algorithms.

Most works on extractors focused on *seeded* extractors, which can utilize an additional seed to aid the extraction. There has been a long and fruitful line of results on constructing seeded extractors (see [23] for a nice survey), which culminated in [19] with an optimal construction (up to constant factors).

However, there is an issue of using seeded extractors. Namely, we need a seed which is perfectly random and independent of the source we extract from. How do we get such a seed? For some applications, this can be taken care of (e.g. by enumerating through all possible seed values), but for others, this seems to go back to the problem which we try to solve using extractors. Can we get rid of the need for a seed and have *seedless* extractors? For general sources, the answer has been known to be negative [6]. However, when the sources are restricted and have special structures, seedless extraction becomes possible. Examples of such sources include samplable sources [26], bit-fixing sources [7,9,16], independent-symbol sources [15,17], and multiple independent sources [1,5,6,2,21,22].

In this paper, we would like to look for a more general class of sources from which seedless extraction is still possible. In particular, we will consider sources which may contain no randomness at all in a statistical sense, but *look* slightly random to computational-bounded observers, such as small circuits. That is, we will go from a traditional, statistical setting to a computational one. It is conceivable that in many situations when we consider a source random, it may in fact only appear so to us, while its actual statistical min-entropy may be much smaller (or even zero) especially if we take into account some correlated information which we can observe. Another application of this notion is in cryptography, and in fact the idea of extracting computational randomness has appeared implicitly long ago since [27,10,12], for the task of constructing pseudo-random generators from one-way functions. The idea is that given a one-way function g , it is hard to invert $g(y)$ to get y , and this means that given the (correlated) information $g(y)$, y still looks somewhat random, from which one can extract some bits that look almost random. However, while there is a natural and well-accepted definition for what we mean that a distribution looks almost random [27], it seems less clear for what we mean that a distribution looks slightly random and for how to measure the amount of randomness in it. In fact, there are several alternatives which all seem reasonable, but there are provable discrepancies among them [3,13]. To extract randomness from a source with so-called HILL-entropy [3], the strongest among them, one can simply use any statistical extractor. Here we consider a weaker (more general) notion of computational randomness, which appears in [13], and we call it *computational min-entropy*.

Computational min-entropy. To model the more general situation that one may observe some correlated information about the sources, we consider sources of a conditional form $(\mathcal{V}|\mathcal{X})$, where $\mathcal{V}$ is the source from which we want to extract and $\mathcal{X}$ (could be empty) is some distribution which one can observe. The correlation between $\mathcal{V}$ and $\mathcal{X}$ is modeled by $\mathcal{V} = f(\mathcal{X})$ for some function f . In the example of one-way function, f is the inverse function g^{-1} , which is hard to compute, and $\mathcal{X}$ is the distribution of $g(y)$ over a random y . Here in our definition, we allow f to be probabilistic and we even do not require it to have an efficient (or even computable) algorithm, and furthermore, we do not require $\mathcal{X}$ to be efficiently samplable either. We say that such a distribution $(f(\mathcal{X})|\mathcal{X})$ has computational min-entropy k if given input x sampled from $\mathcal{X}$, any circuit of size 2^k can only

predict $f(x)$ correctly with probability at most 2^{-k} (a more general definition is to have the circuit size as a separate parameter, but our extractor construction does not seem to work for this general definition). From the distribution $f(\mathcal{X})$, we would like to extract randomness which when given $\mathcal{X}$ still looks random to circuits of a certain size. Note that a source $\mathcal{Y}$ with statistical min-entropy k can be seen as such a source $(f(\mathcal{X})|\mathcal{X})$ with computational min-entropy k , where we can simply have no $\mathcal{X}$ or just have $\mathcal{X}$ taking a fixed value, and let f be a probabilistic function with $\mathcal{Y}$ as its output distribution. This means that extractors for sources with computational min-entropy can immediately work for sources with statistical min-entropy, and thus results in the computational setting can be seen as a generalization of those in the traditional, statistical setting. On the other hand, for a deterministic function f , $f(x)$ has no statistical min-entropy at all when given x . Still, according to our definition, as long as f is hard to compute, $(f(\mathcal{X})|\mathcal{X})$ in fact can have high computational min-entropy.

Extractors for such sources were implicitly proposed before [10,12], and they are seeded ones. In fact, any seeded statistical extractor with some additional *reconstruction* property (in the sense of [25]) gives a seeded extractor for such sources [3,24,13]. However, just as in the statistical setting, several natural questions arise in the computational setting too. To extract from such sources, do we really need a seed? Can we use a weaker seed which is only slightly random in a statistical sense, or an even weaker seed which only look slightly random in a computational sense but may contain no randomness in a statistical sense? Seeing the seed as an additional independent source, a general question is: Can we have seedless extractors for multiple independent sources, each with some computational min-entropy? One can see this as a step toward extending the study of multi-source extractors from the traditional, statistical setting to a new, computational setting. One can also see this as providing a finer map for the landscape of statistical extractors, according to the degree of their reconstruction property.

Our results. First, we show that it is impossible to have seedless extractors for one single source, even if the source of length n can have a computational min-entropy as high as $n - 2$ and even if we only want to extract one bit.

Next, we show that with the help of a weak seed, it becomes possible to extract randomness from such sources. We use a two-source extractor of Lee *et al.* [18], denoted as EXT, which takes two input strings $v, w \in \{0, 1\}^n$, sees them as vectors from $\mathbb{F}^\ell$, where $\mathbb{F} = GF(2^m)$ for some m with $n = m\ell$, and outputs their inner product over $\mathbb{F}$. As shown in [18], it works for any two independent sources both containing some statistical min-entropy. Moreover, it is also known to work when one source contains some computational min-entropy and the other, the seed, is perfectly random (in a statistical sense) [11]. Our second result shows that it even works when the seed only contains some statistical min-entropy. More precisely, we show that given any source $(f(\mathcal{X})|\mathcal{X})$ with computational min-entropy $k_1 = n - k + O(k/\log k)$ and another independent source $\mathcal{W}$ with statistical min-entropy k , $\text{EXT}(f(\mathcal{X}), \mathcal{W})$ given $\mathcal{X}$ cannot be distinguished from random with advantage $\varepsilon = 2^{-O(\sqrt{k/\log k})}$ by circuits of size $s = 2^{n-k+O(k/\log k)}$. Then we proceed to show that it works even when the seed only contains

computational min-entropy. More precisely, for a source $(g(\mathcal{Y})|\mathcal{Y})$ with computational min-entropy k , $\text{EXT}(f(\mathcal{X}), g(\mathcal{Y}))$ given $(\mathcal{X}, \mathcal{Y})$ still cannot be distinguished with advantage ε by circuits of size about s . This can be seen as a seedless extractor for two independent sources, both with computational min-entropy.

We do not know if the statistical extractors of [1,2,22,5,21] for multiple independent sources can work in the computational setting, since to work in this setting, we need them to have some reconstruction property. For the extractors from [10,11], this property can be translated to a task in learning theory, and the proofs there can be recast as providing an algorithm for learning linear functions under *uniform* distribution with adversarial noise. Our second result can be seen as a generalization of [10,11], and we are facing a more challenging learning problem: learning linear functions under *arbitrary* distribution with adversarial noise. Our third result provides an algorithm for this problem, which, in addition to being used to prove our second result, may have interest of its own.

In the learning problem, there is an unknown linear function $v : \mathbb{F}^\ell \rightarrow \mathbb{F}$ which we want to learn, and a distribution $\mathcal{W}$ over $\mathbb{F}^\ell$ from which we can sample w to obtain a training example $(w, q(w))$, for some function $q : \mathbb{F}^\ell \rightarrow \mathbb{F}$. The function q can be seen as a noisy version of v with some noise rate α , and there are two noise models. In the adversarial-noise model, q is a deterministic function such that $\Pr_{w \in \mathcal{W}}[q(w) \neq v(w)] \leq \alpha$. In the random-noise model, q is a probabilistic function such that independently for any w , $\Pr[q(w) \neq v(w)] \leq \alpha$. We consider the more difficult adversarial-noise model, and our algorithm works for an arbitrary distribution $\mathcal{W}$, while its complexity depends on the min-entropy k of $\mathcal{W}$. More precisely, our algorithm samples $2^{O(k/\log k)}$ examples, runs in time $2^{n-k+O(k/\log k)}$, and with high probability outputs a list containing every linear function v satisfying $\Pr_{w \in \mathcal{W}}[q(w) \neq v(w)] \leq \alpha$, for $\alpha = 1 - 2^{-O(\sqrt{k/\log k})}$. The factor 2^{n-k} in our running time is in fact unavoidable because one can easily find a distribution $\mathcal{W}$ for which the number of such v 's, and thus the running time, is in fact at least 2^{n-k} . Note that when $\mathcal{W}$ is the uniform distribution (with $k = n$), our algorithm runs in time $2^{O(n/\log n)}$ and takes $2^{O(n/\log n)}$ samples.

Previously, the algorithm of Blum *et al.* [4] can learn under arbitrary distribution but in the random-noise model, while that of Feldman *et al.* [8] can learn in the adversarial-noise model but under the uniform distribution. Both algorithms learn the parity functions on n variables, tolerate a noise rate $\alpha \leq 1/2 - \Omega(1)$, run in time $2^{O(n/\log n)}$, and take $2^{O(n/\log n)}$ samples. Very recently, Kalai *et al.* [14] gave an algorithm which can learn the parity functions under arbitrary distribution in the adversarial-noise model, but the hypothesis they produce is not in the linear form, so it cannot be used for our extractors. Furthermore, they only produce one hypothesis instead of all the legitimate ones, and their technique does not seem to generalize from the parity functions to the linear functions over larger fields. Thus, to the best of our knowledge, the task our learning algorithm achieves has not been accomplished before. Finally, just as the result of [10] can yield a list-decoding algorithm for Hadamard codes, so can ours, while that of [14] can not. In fact, our list-decoding algorithm can work even when all but 2^k symbols from the codeword are erased and an α fraction

of the remaining symbols are corrupted. It can also be seen as list-decoding a *punctured* Hadamard code, where a punctured code is obtained from a code by deleting all but a small number of symbols from the codeword.

2 Preliminaries

For any $m \in \mathbb{N}$, let $\mathcal{U}_m$ denote the uniform distribution over $\{0, 1\}^m$. Let $\text{SIZE}(s)$ be the class of functions computable by Boolean circuits of size s . We say that a function $D : \{0, 1\}^n \rightarrow \{0, 1\}$ is an ε -distinguisher for two distributions $\mathcal{X}$ and $\mathcal{Y}$ over $\{0, 1\}^n$ if $|\Pr[D(\mathcal{X}) = 1] - \Pr[D(\mathcal{Y}) = 1]| > \varepsilon$. All logarithms in this paper will have base two.

We consider two types of min-entropy: *statistical* min-entropy and *computational* min-entropy. The notion of statistical min-entropy is a standard one, usually just called min-entropy.

Definition 1. We say that a distribution $\mathcal{X}$ has statistical min-entropy k , denoted by $H_\infty(\mathcal{X}) = k$, if for any x , $\Pr[\mathcal{X} = x] \leq 2^{-k}$.

Next, we define the notion of computational min-entropy. Here, we consider conditional distributions of the form $(\mathcal{V}|\mathcal{X})$, with $\mathcal{V} = f(\mathcal{X})$, for some distribution $\mathcal{X}$ and some function f , which could be either probabilistic or deterministic.

Definition 2. We say that a distribution $(\mathcal{V}|\mathcal{X})$ has computational min-entropy k , denoted by $H_c(\mathcal{V}|\mathcal{X}) = k$, if for any $C \in \text{SIZE}(2^k)$, $\Pr[C(\mathcal{X}) = \mathcal{V}] \leq 2^{-k}$.

We consider three kinds of extractors: *statistical* extractors, *hybrid* extractors and *computational* extractors. The notion of statistical extractors is a standard one, usually just called extractors, while we introduce the notions of hybrid extractors and computational extractors.

Definition 3. A function $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^m$ is called a

- (k_1, k_2, ε) -statistical-extractor if for any source $\mathcal{V}$ with $H_\infty(\mathcal{V}) \geq k_1$ and any source $\mathcal{W}$, independent of $\mathcal{V}$, with $H_\infty(\mathcal{W}) \geq k_2$, there is no ε -distinguisher (without any complexity bound) for the distributions $(\mathcal{W}, \text{EXT}(\mathcal{V}, \mathcal{W}))$ and $(\mathcal{W}, \mathcal{U}_m)$.
- $(k_1, k_2, \varepsilon, s)$ -hybrid-extractor if for any source $(\mathcal{V}|\mathcal{X})$ with $H_c(\mathcal{V}|\mathcal{X}) \geq k_1$ and any source $\mathcal{W}$, independent of $(\mathcal{V}|\mathcal{X})$, with $H_\infty(\mathcal{W}) \geq k_2$, there is no ε -distinguisher in $\text{SIZE}(s)$ for the distributions $(\mathcal{X}, \mathcal{W}, \text{EXT}(\mathcal{V}, \mathcal{W}))$ and $(\mathcal{X}, \mathcal{W}, \mathcal{U}_m)$.
- $(k_1, k_2, \varepsilon, s)$ -computational-extractor if for any source $(\mathcal{V}|\mathcal{X})$ with $H_c(\mathcal{V}|\mathcal{X}) \geq k_1$ and any source $(\mathcal{W}|\mathcal{Y})$, independent of $(\mathcal{V}|\mathcal{X})$, with $H_c(\mathcal{W}|\mathcal{Y}) \geq k_2$, there is no ε -distinguisher in $\text{SIZE}(s)$ for the distributions $(\mathcal{X}, \mathcal{Y}, \mathcal{W}, \text{EXT}(\mathcal{V}, \mathcal{W}))$ and $(\mathcal{X}, \mathcal{Y}, \mathcal{W}, \mathcal{U}_m)$.

Note that the definition above corresponds to the notion of strong extractors in the setting of seeded statistical extractors, which guarantees that even given the seed (the second source), the output still looks random.

We will need the following statistical extractor from [18]. For any $m \in \mathbb{N}$ with $m|n$, let $\ell = n/m$, and see any $x \in \{0,1\}^n$ as an ℓ -dimensional vector $x = (x_1, x_2, \dots, x_\ell)$ over $\mathbb{F} = GF(2^m)$. Then for any $x, y \in \mathbb{F}^\ell$, let $\langle x, y \rangle$ be their inner product over $\mathbb{F}$ defined as $\langle x, y \rangle = \sum_{i=1}^\ell x_i \cdot y_i$.

Theorem 1. [18] *The function $\text{EXT} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^m$ defined as $\text{EXT}(u, v) = \langle u, v \rangle$ is a (k_1, k_2, ε) -statistical-extractor when $k_1 + k_2 \geq n + m + 2 \log(1/\varepsilon) - 2$.*

We will need the following lemma (the proof is omitted due to the page limit).

Lemma 1. *Let $\text{EXT} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^m$ be any (k_1, k_2, ε) -statistical-extractor. Then for any source $\mathcal{W}$ over $\{0,1\}^n$ with $H_\infty(\mathcal{W}) = k_2$ and any function $q : \{0,1\}^n \rightarrow \{0,1\}^m$, there are at most 2^{k_1} different v 's satisfying $\Pr_{w \in \mathcal{W}}[q(w) = \text{EXT}(v, w)] \geq 1/2^m + \varepsilon$.*

Finally, we will need the following lemma about obtaining predictors from distinguishers. The Boolean case ($m = 1$) is well known, and a proof for general m can be found in [11].

Lemma 2. *For any source $\mathcal{Z}$ over $\{0,1\}^n$ and any function $b : \{0,1\}^n \rightarrow \{0,1\}^m$, if there is an ε -distinguisher D for the distributions $(\mathcal{Z}, b(\mathcal{Z}))$ and $(\mathcal{Z}, \mathcal{U}_m)$, then there is a predictor P with D as oracle which calls D once and runs in time $O(m)$ such that $\Pr_{z \in \mathcal{Z}}[P^D(z) = b(z)] \geq (1 + \varepsilon)/2^m$.*

3 An Impossibility Result

Just as in the statistical setting [6], we show that seedless extractors do not exist either in the computational setting. In fact, we show the impossibility result even for sources with a computational min-entropy as high as $n - 2$.

Theorem 2. *For any $n_1, n \in \mathbb{N}$ with $n_1 \geq 3n$ and for any function $\text{EXT} : \{0,1\}^n \rightarrow \{0,1\}$, there exists a deterministic function $f : \{0,1\}^{n_1} \rightarrow \{0,1\}^n$ such that $H_c(f(\mathcal{X})|\mathcal{X}) = n - 2$ for $\mathcal{X} = \mathcal{U}_{n_1}$ but $\text{EXT}(f(x))$ takes the same value for all x (so can be easily distinguished from random).*

The proof uses a standard probabilistic method, which we omit here.

4 Hybrid and Computational Extractors

In this section, we show that the function $\text{EXT} : \mathbb{F}^\ell \times \mathbb{F}^\ell \rightarrow \mathbb{F}$ defined in Theorem 1 as $\text{EXT}(v, w) = \langle v, w \rangle$, which is known to be a good statistical extractor, is also a good hybrid extractor and a good computational extractor.

Theorem 3. *For any $k \geq \Omega(\log^2 n)$, any $m \leq O(\sqrt{k/\log k})$ dividing n , any $\varepsilon \geq 2^{-O(\sqrt{k/\log k})}$, any $s \leq 2^{n-k+O(k/\log k)}$, and for some $k_1 = n - k + O(k/\log k)$, the function $\text{EXT} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^m$ defined above is both a (k_1, k, ε, s) -hybrid-extractor and a (k_1, k, ε, s) -computational-extractor.*

The proof for Theorem 3 relies on the following result, which gives an algorithm for the problem of learning linear functions under arbitrary distribution with adversarial noise.

Theorem 4. *For any $k \geq \Omega(\log^2 n)$, any $m \leq O(k/\log k)$ dividing n , and any $\delta \geq 2^{-O(\sqrt{k/\log k})}$, there exists a learning algorithm A with the following property. Given any source $\mathcal{W}$ over $\{0, 1\}^n = \mathbb{F}^\ell$ with $H_\infty(\mathcal{W}) = k$ and any function $q : \mathbb{F}^\ell \rightarrow \mathbb{F}$, the algorithm A samples $2^{O(k/\log k)}$ training examples from the distribution $(\mathcal{W}, q(\mathcal{W}))$ and then runs in time $2^{n-k+O(k/\log k)}$ to output a list of size $2^{n-k+O(k/\log k)}$ which with probability $1 - o(1)$ contains every $v \in \mathbb{F}^\ell$ satisfying $\Pr_{w \in \mathcal{W}} [q(w) = \langle v, w \rangle] \geq 1/2^m + \delta$.*

Note that as in a standard learning-theoretical setting, we do not count the complexity of sampling the training examples in Theorem 4. We will prove the theorem in the next section, and let us see how it is used to show Theorem 3.

Due to the page limit, we omit the formal proof of Theorem 3 and only sketch the proof idea here. For the case of a hybrid extractor, consider any two independent sources $(\mathcal{V}|\mathcal{X})$ and $\mathcal{W}$, and assume that there is a distinguisher for $(\mathcal{X}, \mathcal{W}, \langle \mathcal{V}, \mathcal{W} \rangle)$ and $(\mathcal{X}, \mathcal{W}, \mathcal{U}_m)$. Then by Lemma 2 and a Markov inequality, there is a predictor Q such that for many (x, v) , $\Pr_{w \in \mathcal{W}} [Q(x, w) = \langle v, w \rangle] \geq 1/2^m + \delta$. For such a (x, v) , we want to predict v from x , which can be seen as learning the linear function $\langle v, \cdot \rangle$ through noisy training examples $(w, q(w))$, with $q(w) = Q(x, w)$, under the distribution $w \in \mathcal{W}$. Using Theorem 4 with the function $q(\cdot) = Q(x, \cdot)$, we can obtain an algorithm to predict v from x with a good probability for these many (x, v) 's. This contradicts the assumption about the computational min-entropy of $(\mathcal{V}|\mathcal{X})$. The proof for computational extractors is almost identical, using the fact that $H_c(\mathcal{W}|\mathcal{Y}) = k$ implies $H_\infty(\mathcal{W}) = k$.

5 Learning Noisy Linear Functions

In this section, we sketch the proof of Theorem 4. Recall that given any source $\mathcal{W}$ over $\{0, 1\}^n = \mathbb{F}^\ell$ with $H_\infty(\mathcal{W}) = k$, any $\delta \geq 2^{-O(\sqrt{k/\log k})}$, and any function $q : \mathbb{F}^\ell \rightarrow \mathbb{F}$, we would like to learn some unknown $v \in \mathbb{F}^\ell$ such that

$$\Pr_{w \in \mathcal{W}} [q(w) = \langle v, w \rangle] \geq 1/2^m + \delta. \quad (1)$$

Since such v may not be unique, we will list them all. Let us first imagine one such fixed v .

We start by randomly choosing $K = 2^{c(k/\log k)}$ independent training examples (with replacement) from the distribution $(\mathcal{W}, q(\mathcal{W}))$, for some large enough constant c (depending on δ). Let $W^{(0)}$ denote the $K \times \ell$ matrix and $q^{(0)}$ the K -dimensional vector, both over $\mathbb{F}$, such that for each training example $(w, q(w))$, $W^{(0)}$ has $w \in \mathbb{F}^\ell$ as a row and $q^{(0)}$ has $q(w) \in \mathbb{F}$ as an entry. Note that each training example $(w, q(w))$, with $w = (w_1, w_2, \dots, w_\ell)$, gives us a linear equation $w_1 v_1 + w_2 v_2 + \dots + w_\ell v_\ell = q(w)$ for $v = (v_1, v_2, \dots, v_\ell) \in \mathbb{F}^\ell$. Thus from these K training examples, we obtain a system of K linear equations, denoted as $[W^{(0)}|q^{(0)}]$, and we would like to reduce the task of learning v to that of solving this system of linear equations. However, this system is highly noisy as about $1 - 1/2^m$ fraction of the equations are likely to be wrong, according to (1). We will roughly follow the approach of Gaussian Elimination (which works

1. For t from 1 to T do
 - (a) Partition the equations of $[W^{(t-1)}|q^{(t-1)}]$ into at most 2^{md} groups according to their first blocks in $W^{(t)}$ (same block value in the same group).
 - (b) Within each group, randomly select an equation which we call pivot.
 - (c) Within each group, subtract each equation by the pivot.
 - (d) Remove the pivots and delete the first block from each equation. Let $[W^{(t)}|q^{(t)}]$ be the resulting system of equations.

Fig. 1. FORWARD PHASE

1. Set $V^{(T)} = \mathbb{F}^{(n-k)/m}$, and set $V^{(t)} = \emptyset$ for $0 \leq t \leq T - 1$.
2. For t from $T - 1$ down to 0 do
 - (a) For any $z \in \mathbb{F}^d \times V^{(t+1)}$ which is δ_t -good for $[W^{(t)}|q^{(t)}]$, include z into $V^{(t)}$ if $|V^{(t)}| \leq L$, and break otherwise.
3. Output $V^{(0)}$.

Fig. 2. BACKWARD PHASE

for noiseless systems of linear equations), but will make substantial changes in order to deal with our noisy case.

Our algorithm consists of two phases: the forward phase, shown in Figure 1, and the backward phase, shown in Figure 2. The forward phase works as follows, which is similar to the approach of Blum *et al.* [4]. Starting from the system $[W^{(0)}|q^{(0)}]$, we use several iterations to produce smaller and smaller systems with fewer and fewer variables, until we have a system which we can afford to solve using brute force. More precisely, we choose the parameters

$$T = \log \sqrt{k/\log k} \quad \text{and} \quad d = k/(mT),$$

divide each row of $W^{(0)}$ into ℓ/d blocks, with each block containing d elements in $\mathbb{F}$, and proceed in T iterations, as shown in Figure 1. Note that after iteration t , we have the system $[W^{(t)}|q^{(t)}]$ which has $\ell - dt$ variables and $K^{(t)}$ equations, with $K^{(t)} \geq K - t2^{md} \geq 2^{c(k/\log k)}/2 = K/2$, for a large enough constant c . The key is to guarantee that the system still contains a good fraction of correct equations. Let $\delta_0 = \delta/2$ and $\delta_t = (\delta_{t-1}/2)^2$ for $t \geq 1$, and a simple induction shows that $\delta_t \geq (\delta/8)^{2^t} \geq 2^{-0.1c(k/\log k)} = K^{-0.1}$, for a large enough constant c . We say that any $z \in \mathbb{F}^{\ell-dt}$ is δ_t -good for the system $[W^{(t)}|q^{(t)}]$ if it satisfies at least $1/2^m + \delta_t$ fraction of equations in the system. Let $v^{(t)} \in \mathbb{F}^{\ell-dt}$ denote v without its first t blocks, and we call the forward phase *good* if for every t , $v^{(t)}$ is δ_t -good for $[W^{(t)}|q^{(t)}]$. Lemma 3 below, whose proof is omitted, guarantees that the forward phase is good with a significant probability.

Lemma 3. *The forward phase is good with probability at least $2^{-O(k/\log k)}$.*

For the backward phase, we start from the last system $[W^{(T)}|q^{(T)}]$ produced by the forward phase, and work backward on larger and larger systems produced in the forward phase to obtain solutions for more and more variables. More precisely, we go from $t = T - 1$ down to $t = 0$, and while in iteration t , we try to find all possible solutions which extend solutions from iteration $t + 1$ and

are δ_t -good for $[W^{(t)}|q^{(t)}]$, as shown in Figure 2. However, in order to bound the running time, we will stop including the solutions once the number grows beyond the threshold

$$L = 2^{n-k+m+T+2\log(1/\delta_T)} \leq 2^{n-k+O(k/\log k)}.$$

If this happens, we may fail to include the actual solution v in our final list. Call the backward phase *good* if for every t , the number of such δ_t -good solutions for $[W^{(t)}|q^{(t)}]$ is at most L . Lemma 4 below guarantees that the backward phase is indeed good with a high probability.

Lemma 4. *The backward phase is not good with probability at most $2^{-\Omega(k)}$.*

To get an idea of the proof, let's consider how to bound the number of δ_0 -good solutions for the system $[W^{(0)}|q^{(0)}]$. Since EXT is a good *statistical* extractor and $\mathcal{W}$ has a high min-entropy, Lemma 1 gives us a bound on the number of z satisfying $\Pr_{w \in \mathcal{W}}[q(w) = \langle z, w \rangle] \geq 1/2^m + \delta_0/2$. Then as each row of $W^{(0)}$ is sampled independently from $\mathcal{W}$, one can show that any other z not satisfying the probability bound is very unlikely to be δ_0 -good for $[W^{(0)}|q^{(0)}]$ by a Chernoff bound. Now for $t \geq 1$, to follow this idea to bound the number of good solutions for $[W^{(t)}|q^{(t)}]$, we would also like the rows of $W^{(t)}$ to come independently from a high min-entropy source. However, this may not be true in general. Our approach is to fix the set of pivots in the first t iterations (leaving all other rows free) and show that the resulting distribution of $W^{(t)}$ conditioned on the fixing is close to having this desirable property. Due to the page limit, we omit the proof here.

Let us measure the complexity of our algorithm. First, $K \leq 2^{O(k/\log k)}$ training examples are sampled from the distribution $(\mathcal{W}, q(\mathcal{W}))$. Then the forward phase runs in time $T \cdot \text{poly}(K)$, and the backward phase runs in time $T \cdot 2^{md} \cdot L \cdot K$, so the total running time is at most $2^{n-k+O(k/\log k)}$.

Finally, we bound the success probability of our algorithm. From Lemma 3 and Lemma 4, the probability that both the forward and backward phases are good is at least $2^{-O(k/\log k)}$. Assuming both phases are good, one can show that $v \in V^{(0)}$. Thus, any fixed v satisfying the bound in (1) is contained in the list $V^{(0)}$ of size at most L with probability $2^{-O(k/\log k)}$. We can further reduce the probability of missing this v to $2^{-\omega(n)}$ by repeating the process $2^{O(k/\log k)}$ times, and take the union of the produced lists. Then a union bound shows that some v satisfying (1) is not included in the final output with probability $o(1)$. Thus, we have Theorem 4.

References

1. Barak, B., Impagliazzo, R., Wigderson, A.: Extracting randomness using few independent sources. In: FOCS 2004, pp. 384–393 (2004)
2. Barak, B., Kindler, G., Shaltiel, R., Sudakov, B., Wigderson, A.: Simulating independence: new constructions of condensers, Ramsey graphs, dispersers, and extractors. In: STOC 2005, pp. 1–10 (2005)
3. Barak, B., Shaltiel, R., Wigderson, A.: Computational analogues of entropy. In: Arora, S., Jansen, K., Rolim, J.D.P., Sahai, A. (eds.) RANDOM 2003 and APPROX 2003. LNCS, vol. 2764, pp. 200–215. Springer, Heidelberg (2003)

4. Blum, A., Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM* 50(4), 506–519 (2003)
5. Bourgain, J.: More on the sum-product phenomenon in prime fields and its applications. *International J. Number Theory* 1, 1–32 (2005)
6. Chor, B., Goldreich, O.: Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Comput.* 17(2), 230–261 (1988)
7. Chor, B., Goldreich, O., Håstad, J., Friedman, J., Rudich, S., Smolensky, R.: The bit extraction problem of t -resilient functions. In: *FOCS 1985*, pp. 396–407 (1985)
8. Feldman, V., Gopalan, P., Khot, S., Ponnuswami, A.K.: New results for learning noisy parities and halfspaces. In: *FOCS 2006*, pp. 563–574 (2006)
9. Gabizon, A., Raz, R., Shaltiel, R.: Deterministic extractors for bit-fixing sources by obtaining an independent seed. In: *FOCS 2004*, pp. 394–403 (2004)
10. Goldreich, O., Levin, L.A.: A hard-core predicate for all one-way functions. In: *STOC 1989*, pp. 25–32 (1989)
11. Goldreich, O., Rubinfeld, R., Sudan, M.: Learning polynomials with queries: the highly noisy case. *SIAM J. Disc. Math.* 13(4), 535–570 (2000)
12. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. *SIAM J. Comput.* 28(4), 1364–1396 (1999)
13. Hsiao, C.-Y., Lu, C.-J., Reyzin, L.: Conditional computational entropy, or toward separating pseudoentropy from compressibility. In: Naor, M. (ed.) *EUROCRYPT 2007*. LNCS, vol. 4515, pp. 169–186. Springer, Heidelberg (2007)
14. Kalai, A., Mansour, Y., Verbin, E.: On agnostic boosting and parity learning. In: *STOC 2008*, pp. 629–638 (2008)
15. Kamp, J., Rao, A., Vadhan, S., Zuckerman, D.: Deterministic extractors for small-space sources. In: *STOC 2006*, pp. 691–700 (2006)
16. Kamp, J., Zuckerman, D.: Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. In: *FOCS 2003*, pp. 92–101 (2003)
17. Lee, C.-J., Lu, C.-J., Tsai, S.-C.: Deterministic extractors for independent-symbol sources. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) *ICALP 2006*. LNCS, vol. 4051, pp. 84–95. Springer, Heidelberg (2006)
18. Lee, C.-J., Lu, C.-J., Tsai, S.-C., Tzeng, W.-G.: Extracting randomness from multiple independent sources. *IEEE Trans. on Info. Theory* 51(6), 2224–2227 (2005)
19. Lu, C.-J., Reingold, O., Vadhan, S.P., Wigderson, A.: Extractors: optimal up to constant factors. In: *STOC 2003*, pp. 602–611 (2003)
20. Nisan, N., Zuckerman, D.: Randomness is linear in space. *J. Comput. Syst. Sci.* 52(1), 43–52 (1996)
21. Rao, A.: Extractors for a constant number of polynomially small min-entropy independent sources. In: *STOC 2006*, pp. 497–506 (2006)
22. Raz, R.: Extractors with weak random seeds. In: *STOC 2005*, pp. 11–20 (2005)
23. Shaltiel, R.: Recent developments in explicit constructions of extractors. *Bulletin of the EATCS* 77, 67–95 (2002)
24. Ta-Shma, A., Zuckerman, D.: Extractor codes. *IEEE Trans. Info. Theory* 50(12), 3015–3025 (2004)
25. Trevisan, L.: Extractors and pseudorandom generators. *J. ACM* 48(4), 860–879 (2001)
26. Trevisan, L., Vadhan, S.: Extracting randomness from samplable distributions. In: *FOCS 2000*, pp. 32–42 (2000)
27. Yao, A.C.: Theory and applications of trapdoor functions. In: *FOCS 1982*, pp. 80–91 (1982)
28. Zuckerman, D.: General weak random sources. In: *FOCS 1990*, pp. 534–543 (1990)

HITS Can Converge Slowly, but Not Too Slowly, in Score and Rank

Enoch Peserico* and Luca Pretto

Dip. Ing. Informazione, Univ. Padova, Italy
{`enoch,pretto`}@dei.unipd.it

Abstract. This paper explores the fundamental question of how many iterations the celebrated HITS algorithm requires on a general graph to converge in score and, perhaps more importantly, in rank (i.e. to “get right” the order of the nodes). We prove upper and almost matching lower bounds. We also extend our results to weighted graphs.

1 Introduction

How many iterations does HITS require on a general graph to converge in score and, perhaps more importantly, in rank? This introduction motivates the question (subsection 1.1), and provides a brief description of our results and of the organization of the rest of the paper (subsection 1.2).

1.1 Motivation and Related Work

Kleinberg’s celebrated HITS (Hypertext Induced Topic Search) algorithm [11] is one of the most famous link analysis algorithms and probably *the* most widely used outside the context of Web search - making it a reference algorithm for today’s link analysis, much like quicksort or heapsort for sorting. HITS was originally proposed to rank Web pages, and is the basis of search engines such as Ask.com. It has been subsequently employed, sometimes with small variations, to rank graph nodes in a vast and growing number of application domains, often with little or no connection to Web search (e.g. [3,2,10,14,19,23]).

HITS is an iterative algorithm computing for each node of a generic graph an *authority* score at every iteration. Thus, any analysis of its computational requirements must consider the number of iterations required to converge within a sufficiently small distance of a limit score vector (which always exists, [11]). In fact, most applications employ the score vector directly to *rank* the nodes of the target graph. In all these cases it is even more important to understand the number of iterations required by HITS to converge *in rank* - informally to assign scores to nodes, that could be potentially quite different from the limit scores, but that still place all or almost all nodes in the “correct” order. The issue of rank convergence as opposed to score convergence is indeed widely regarded as one of the major theoretical challenges of link analysis [8,13,16,20].

* Supported in part by EU under Integr. Proj. AEOLUS (IP-FP6-015964).

HITS effectively computes the dominant eigenvector of $\mathbf{A}^T \mathbf{A}$ (where $\mathbf{A}$ is the adjacency matrix of the input graph) using the Power Method [7] and thus its speed of convergence in score is tied to the separation of the first and second eigenvalues of that matrix. However, no bounds on this separation are known *for the matrix derived from a graph* - for arbitrary matrices of any fixed size the separation can be arbitrarily small and the convergence rate arbitrarily slow. Perhaps even more importantly, no bounds are known on the convergence of HITS *in rank*: only a few experimental results are available, and only for the Web Graph [11]. Being heavily application-dependent, these provide little information for porting the famous algorithm to new application domains.

1.2 Our Results

This is the first paper providing non-trivial bounds on the convergence rate of HITS, both in score and in rank (some less general lower bounds can be found in our technical report [21]). In a nutshell, we show that HITS can converge slowly, but not too slowly, both in score and in rank. On the one hand, we show that an exponential number of iterations might be necessary to converge to a ranking (and a score) that is even remotely accurate. On the other hand, we show that an exponential number of iterations is also sufficient - and since the iterative process can be accelerated through a well-known “repeated squaring trick” [12] this entails that the complexity of converging to a result extremely close to the limit ranking (and score) is at most polynomial in the number of nodes n of the graph ($O(n^{4+\mu} \lg(n))$, where $\Theta(n^{2+\mu})$ is the complexity of n by n matrix multiplication). Thus, acceleration by repeated squaring seems both sufficient and, in general, necessary to provide convergence guarantees for HITS on graphs of moderate size. The rest of the paper is organized as follows.

Section 2 briefly reviews HITS and some well-known results tying the convergence of the Power Method to the eigengap of a matrix.

Section 3 formally defines the apparently natural, but extremely slippery notion of convergence in rank (for a more detailed discussion of the topic, see [20]).

Section 4 exploits the structure of the matrix $\mathbf{A}^T \mathbf{A}$ to provide bounds on the separation of its eigenvalues, and translates those bounds into upper bounds on the convergence of the score vector provided by HITS. In particular, we show that HITS never requires more than $(\lg(\frac{1}{\epsilon}) + \lg(n))(wg)^{\Theta(m^2)}$ iterations to converge, both in score and in rank, to a vector within distance ϵ of the limit score vector, on any n -node graph of maximum degree g whose links have integer weights bounded by w , and whose authority connected components [18] have at most $m \leq n$ nodes. We tighten this bound to $(\lg(\frac{1}{\epsilon}) + \lg(n))(wg)^{\Theta(m)}$ iterations when the dominant eigenvalues of $\mathbf{A}^T \mathbf{A}$ belong to the same irreducible block - this includes the important class of authority connected graphs [18]. In this case, the integrality condition can also be relaxed into requiring minimum weight 1.

These bounds are not as weak as they might appear, for two reasons. First of all, note that one might compute the p^{th} power of an n by n matrix $\mathbf{M}$ with at most $2\lceil \lg(p) \rceil$ matrix multiplications using a “repeated squaring trick” - first computing $\mathbf{M}, \mathbf{M}^2, \mathbf{M}^4, \dots, \mathbf{M}^{2^{\lceil \lg(p) \rceil}}$ and then multiplying an appropriate subset

of those $\lfloor \lg(p) \rfloor$ matrices [12]. Thus, our upper bounds show that, on any n node graph, the complexity of computing the HITS score vector to any precision up to $\frac{1}{2^{2^{\Theta(n)}}}$ is $O(n^{4+\mu} \lg(n))$ - where $n^{2+\mu}$ with $0 \leq \mu \leq 1$ is the complexity of n by n matrix multiplication - and $O(n^{3+\mu} \lg(n))$ in the case of authority connected graphs. This holds even if the arcs have integer weights bounded by $\text{poly}(n)$.

Furthermore, Section 5 almost matches the upper bounds of Section 4 by exhibiting, for all $s \geq 3$, unweighted authority connected graphs of maximum degree k and $\approx k^3 + 3ks$ nodes that, even after $k^{\Theta(s)}$ iterations, fail to “get right” more than $k + 1$ of the top $k^2 + k$ ranks, even if one accepts as “correct” the ranking provided not just by the limit score vector, but by any score vector at distance less than $\bar{\epsilon} = \Theta(\frac{1}{k\sqrt{k}})$ from it. This also implies lack of convergence to a distance less than $\bar{\epsilon}$ of the limit score vector. In other words, HITS fails not only to get the score error below a relatively large value (since the score vector of HITS is normalized in $\|\cdot\|_2$, the $(k^2)^{\text{th}}$ largest component itself can be no larger than $1/k$), but fails also to “get right” more than a small fraction of the top ranks unless allowed to run for exponentially many iterations.

Section 6 summarizes our results, discusses their significance, and briefly reviews some problems left open - before concluding with the bibliography.

2 HITS

This section briefly describes the original HITS algorithm designed for the Web graph [11] (subsection 2.1) summarizing the most important mathematical results in the literature (subsection 2.2, see [9,1,6] for more details). Many pre-processing heuristics, typically dependent on the application domain, have been proposed to modify the target graph (e.g. the removal of all intradomain links as biased conferrals of authority [11]). One should then interpret our analysis as applying to the resulting graph.

2.1 The Algorithm

The original version of HITS works as follows. In response to a query, a search engine first retrieves a set of nodes of the Web graph on the basis of pure textual analysis; for each such node it also retrieves all nodes pointed by it, and up to d nodes pointing to it. HITS operates on the subgraph induced by this *base set* (which obviously depends on the query) associating to each node v_i an *authority* score a_i that summarizes both its quality and its relevance to the query, as well as an ancillary *hub* score h_i , according to the iterative formulas:

$$h_i^{(0)} = 1 \quad a_i^{(t)} = \sum_{v_j \rightarrow v_i} h_j^{(t-1)} \quad h_i^{(t)} = \sum_{v_i \rightarrow v_j} a_j^{(t)}, \quad t = 1, 2, \dots$$

where $v \rightarrow u$ denotes that v points to u .

More precisely, at each step the authority and hub vector of scores are normalized in $\|\cdot\|_2$ (this is well defined assuming, as we shall do throughout the

rest of the paper, that the subgraph induced by the base set has at least one arc). Then the authority vector $\mathbf{a}^{(t+1)}$, whose i^{th} element $a_i^{(t+1)}$ corresponds to the authority of node i at timestep $t + 1$, can be computed from the adjacency matrix $\mathbf{A}$ of the base set subgraph:

$$\mathbf{a}^{(t+1)} = \frac{(\mathbf{A}^T \mathbf{A})^t \mathbf{A}^T \mathbf{1}}{\|(\mathbf{A}^T \mathbf{A})^t \mathbf{A}^T \mathbf{1}\|_2} \quad (1)$$

where $\mathbf{1}$ is the vector $[1 \dots 1]^T$.

2.2 Convergence of the Authority Vector

Equation (1) shows that HITS is essentially computing a dominant eigenvector of $\mathbf{A}^T \mathbf{A}$ using the iterative Power Method ([7]) starting from the initial vector $\mathbf{A}^T \mathbf{1}$; since $\mathbf{A}^T \mathbf{A}$ is symmetric and positive semidefinite the convergence to a limit vector is guaranteed ([7]).

It is well known, and easy to verify, that the error of approximating the limit vector with $\mathbf{a}^{(t)}$ is tied both to the gap between the largest and second largest eigenvalues of $\mathbf{A}^T \mathbf{A}$, and to the modulus of the projection of the initial vector $\mathbf{A}^T \mathbf{1}$ on the dominant eigenspace.

Since $\mathbf{A}^T \mathbf{A}$ is symmetric and positive semidefinite, its eigenvectors form an orthonormal base and its eigenvalues are all real and non-negative. Denote by $\lambda_1, \dots, \lambda_m$ the m distinct eigenvalues of $\mathbf{A}^T \mathbf{A}$, with $\lambda_i > \lambda_{i+1}$. Then one can write:

$$\mathbf{A}^T \mathbf{1} = \alpha_1 \mathbf{v}_1 + \dots + \alpha_m \mathbf{v}_m$$

where $\mathbf{v}_i$ is a normalized eigenvector relative to λ_i , and $\alpha_1 > 0$ ([1]). Therefore

$$\frac{(\mathbf{A}^T \mathbf{A})^t \mathbf{A}^T \mathbf{1}}{\|(\mathbf{A}^T \mathbf{A})^t \mathbf{A}^T \mathbf{1}\|_2} = \frac{\mathbf{v}_1 + \frac{\alpha_2 \lambda_2^t}{\alpha_1 \lambda_1^t} \mathbf{v}_2 + \dots + \frac{\alpha_m \lambda_m^t}{\alpha_1 \lambda_1^t} \mathbf{v}_m}{\|\mathbf{v}_1 + \frac{\alpha_2 \lambda_2^t}{\alpha_1 \lambda_1^t} \mathbf{v}_2 + \dots + \frac{\alpha_m \lambda_m^t}{\alpha_1 \lambda_1^t} \mathbf{v}_m\|_2}$$

and the last term obviously converges to $\mathbf{v}_1$ since $\lim_{t \rightarrow \infty} \frac{\alpha_i \lambda_i^t}{\alpha_1 \lambda_1^t} = 0$ for $i > 1$.

The 2-norm of the error vector $\bar{\mathbf{v}}^t$ after t steps (assuming $\mathbf{A}$ has order $n \geq m$ and none of its columns adds up to more than r) is then equal to:

$$\begin{aligned} \|\bar{\mathbf{v}}^t\|_2 &= \left\| \frac{\mathbf{v}_1 + \frac{\alpha_2 \lambda_2^t}{\alpha_1 \lambda_1^t} \mathbf{v}_2 + \dots + \frac{\alpha_m \lambda_m^t}{\alpha_1 \lambda_1^t} \mathbf{v}_m}{\|\mathbf{v}_1 + \frac{\alpha_2 \lambda_2^t}{\alpha_1 \lambda_1^t} \mathbf{v}_2 + \dots + \frac{\alpha_m \lambda_m^t}{\alpha_1 \lambda_1^t} \mathbf{v}_m\|_2} - \mathbf{v}_1 \right\|_2 \\ &= \left\| \frac{\mathbf{v}_1 + \sum_{i=2}^m \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1}\right)^t \mathbf{v}_i - (1 + \sum_{i=2}^m \left(\frac{\alpha_i}{\alpha_1}\right)^2 \left(\frac{\lambda_i}{\lambda_1}\right)^{2t})^{\frac{1}{2}} \mathbf{v}_1}{(1 + \sum_{i=2}^m \left(\frac{\alpha_i}{\alpha_1}\right)^2 \left(\frac{\lambda_i}{\lambda_1}\right)^{2t})^{\frac{1}{2}}} \right\|_2 \\ &\leq (2 \sum_{i=2}^m \left(\frac{\alpha_i}{\alpha_1}\right)^2 \left(\frac{\lambda_i}{\lambda_1}\right)^{2t})^{\frac{1}{2}} \leq \frac{(2n)^{\frac{1}{2}}}{\alpha_1} \left(\frac{\lambda_2}{\lambda_1}\right)^t r \end{aligned} \quad (2)$$

where the last inequality follows from the fact that $(\sum_{i=1}^m \alpha_i^2)^{\frac{1}{2}} = \|\mathbf{A}^T \mathbf{1}\|_2 \leq r \|\mathbf{1}\|_2 = rn^{1/2}$. It may then seem that the convergence of HITS is well understood; this is not the case, as we shall see in the next sections.

3 Convergence in Score vs. Convergence in Rank

All the most popular link analysis algorithms iteratively compute a score for each node of the input graph. In many applications (e.g. [4,2,11,17]) this score vector is used alone to rank the nodes - whether because no other scores might be reasonably combined with it (e.g. in Web crawling, word stemming, automatic construction of summaries), or because the algorithm operates on a query dependent graph already capturing the relevance of each node to the query at hand (as in the case of HITS as opposed to e.g. PageRank). In these cases the speed of convergence in score of the algorithm is less crucial than the speed of *convergence in rank* - informally how many iterations are required to rank the nodes in the “correct” order (that induced by the limit score vector).

This informal definition suffers from at least two major flaws. First, it fails to explicitly deal with ties or “almost ties” in score: if the difference between the limit scores of two nodes is negligible, an algorithm effectively converges to a “correct” ranking even if it keeps switching the relative positions of the two nodes. Second, the definition above fails to distinguish between an algorithm that takes a long time to reach the ultimate ranking, but quickly reaches a ranking “close” to it (e.g. with all elements correctly ranked, save the last few), and an algorithm that simply fails for a long time to produce a ranking even remotely close to the ultimate ranking. In this regard, the top ranks are typically much more important than the last ones: failing to converge on the last 20 ranks is almost always far less of a problem than failing to converge on the top 20.

To address these two issues we introduce a more general and formal definition of convergence in rank. We first formalize the notion of ranking induced by a score vector:

Definition 1. *Given a score vector $\mathbf{v} = [v_1, \dots, v_n]$, a ranking ρ compatible with $\mathbf{v}$ is an ordered n -tuple $[i_1, \dots, i_n]$ containing each integer between 1 and n exactly once, such that $\forall j \ v_{i_j} \geq v_{i_{j+1}}$.*

Informally, a ranking is compatible with a score vector if no node that has a higher score is ranked worse than one with a lower score (ties can be broken arbitrarily).

Definition 2. *Consider an iterative algorithm ALG producing at each iteration t a score vector $\mathbf{v}(t)$ and converging to a score vector $\mathbf{v}(\infty)$. Then ALG ϵ -converges on h of the top k ranks in (at most) τ steps if, for all iterations $t \geq \tau$, at least h of the top k items in a ranking compatible with $\mathbf{v}(t)$ are also among the top k items in a ranking compatible with $\mathbf{v}(\infty)$, or compatible with some vector $\mathbf{w}(t)$ at distance at most ϵ from $\mathbf{v}(\infty)$.*

In other words, we assume an algorithm has converged in ranking as soon as it “gets right” (and keeps getting right) at least h of the top k items of any ranking compatible with the limit score vector, or with any score vector “sufficiently close” to it (note that the definition above implicitly assumes some distance function between score vectors - e.g. $\|\cdot\|_2$).

Our definition is related to the notion of intersection metric [5]. The distance in the intersection metric of two rankings that share $h(k)$ of the top k items is the average over k of $1 - h(k)/k$. In our definition we do not “summarize” the size of the intersection between the top k ranks of the current and limit rankings, instead leaving k as a parameter. Furthermore, we consider “acceptable” a whole set of limit rankings induced by “sufficiently close” score vectors.

It is important to note that, with $\epsilon = 0$ and $h = k \forall k$ our definition collapses back to the stricter, “naive” definition of convergence in rank; and that if an algorithm ϵ -converges in score in t iterations (i.e. if the score vector after t iterations always remains within distance ϵ of the limit vector) then that algorithm also ϵ -converges on all its ranks (i.e. on k of its top k ranks $\forall k$) in t steps - but the reverse is not necessarily true.

4 Upper Bounds

While HITS can take many iterations to converge in score or rank (see Section 5), in general it can not take *too many*. This section provides upper bounds on this number (in subsection 4.2) exploiting lower bounds on the separation between the eigenvalues of $\mathbf{A}^T \mathbf{A}$ (in subsection 4.1). These separation bounds are yielded by two completely different proof techniques; which is applicable depends on subtle hypotheses both on the graph topology and on the link weights.

4.1 Some Novel Bounds on Eigenvalue Separation

Assume without loss of generality that $\mathbf{A}^T \mathbf{A}$ is a block matrix - this can always be achieved through appropriate row and column transpositions that correspond to a simple renaming of the nodes. The eigenvalue separation bounds in this subsection depend on whether the two largest eigenvalues are dominant eigenvalues of two different blocks of $\mathbf{A}^T \mathbf{A}$, or if one isn’t. In the latter situation (Lemma 2), the bounds are tighter than in the former (Lemma 1). Note that the latter situation includes the important class of *authority connected graphs* [18]: informally, these are graphs where, for every pair of nodes v and v' with positive indegree, one can reach v from v' by first following a link backwards, then following a link forward, then again a link backwards - and so on. The two proofs are omitted due to space limitations but can be found in the full version of the paper [22].

Lemma 1. *Let $\mathbf{B}_1$ and $\mathbf{B}_2$ be two integer, symmetric, non-negative and positive semidefinite m_1 by m_1 and m_2 by m_2 matrices, with no row adding up to more than (respectively) r_1 and r_2 , whose dominant eigenvalues are (respectively) λ_1 and $\lambda_2 < \lambda_1$. Then $\frac{\lambda_1}{\lambda_2} \geq 1 + 2^{1-m_2} (m_2 + 1)^{\frac{1}{2}-m_1} (m_1 + 1)^{-\frac{m_2}{2}} r_1^{-m_1 m_2} r_2^{-m_1 m_2 - 1} = 1 + r_1 r_2^{-\Theta(m_1 m_2)}$.*

Lemma 2. *Let $\mathbf{B}$ be a symmetric, irreducible, positive semidefinite m by m matrix, whose non-zero elements are all at least 1. Denote by λ_1 and λ_2 , respectively, the first and second eigenvalue of $\mathbf{B}$. If $\lambda_2 \neq 0$, then $\lambda_1 \geq \lambda_2 (1 + 2\lambda_2^{-2m})^{\frac{1}{2m}}$.*

4.2 Upper Bounds on the ϵ -Convergence of HITS

Lemmas 1 and 2, as well as Equation (2), allow us to derive upper bounds on the number of iterations required for HITS ϵ -converge in score (in $\|\cdot\|_2$) - and thus on all ranks. In fact, our results are also applicable to weighted graphs, allowing us to deal with modifications of HITS that assign weights to links (e.g. [15,19]) as long as these weights satisfy some mild conditions.

Theorem 1. *Let G be a graph of n nodes and maximum degree g whose arcs have weights at least 1 and at most w . Denoting by $\mathbf{A}$ the weighted adjacency matrix of G , if $\mathbf{A}^T \mathbf{A}$ is a block matrix such that all its non-zero blocks have size at most m and if the largest and the second largest eigenvalues of $\mathbf{A}^T \mathbf{A}$ are relative to the same block (including the case of just one non-zero block, i.e. if G is authority connected), then HITS ϵ -converges in score (in $\|\cdot\|_2$), and therefore on all ranks, on the nodes of G in at most $m(wg)^{4m}(\lg(\frac{1}{\epsilon}) + \frac{1}{2}\lg(2n) + \lg(wg)) = (wg)^{O(m)}(\lg(\frac{1}{\epsilon}) + \lg(n))$ iterations.*

Theorem 2. *Let G be a graph of n nodes and maximum degree g whose arcs have integer weights at least 1 and at most w . Denote by $\mathbf{A}$ the weighted adjacency matrix of G . If $\mathbf{A}^T \mathbf{A}$ is a block matrix with at least two blocks of size m_1 and m_2 whose dominant, positive eigenvalues λ_1 and $\lambda_2 < \lambda_1$ are respectively the largest and second largest eigenvalue of $\mathbf{A}^T \mathbf{A}$, then HITS ϵ -converges in score (in $\|\cdot\|_2$), and therefore on all ranks, on the nodes of G in at most $2^{m_2-1}(m_2+1)^{m_1-\frac{1}{2}}(m_1+1)^{\frac{m_2}{2}}(wg)^{2m_1m_2}(wg)^{(2m_1m_2+2)}(\lg(\frac{1}{\epsilon}) + \frac{1}{2}\lg(2n) + \lg(wg)) = (wg)^{O(m_1m_2)}(\lg(\frac{1}{\epsilon}) + \lg(n))$ iterations.*

Again, we omit the proofs from this extended abstract due to space limitations. However, it is interesting to compare the conditions that Theorems 1 and 2 place on the link weights. Both require links to fall between 1 and some maximum weight w , which is equivalent to requiring a maximum ratio of w between link weights. In addition, Theorem 2 requires the weights to be integers: this enforces a separation between different weights.

It is easy to prove that *the bound on link weights is essential to ensure bounds on convergence, even if a graph is authority connected*. Consider an undirected graph G formed of 6 vertices $v_1, \dots, v_6$, with v_1, v_2 and v_3 forming a 3-clique, v_4 linked (only) to v_1 , v_5 to v_2 , and v_6 to v_3 ; and an isomorphic graph G' , whose generic node v'_i corresponds to v_i , with v_6 connected to v'_6 (see Figure 1). With a weight of ab on all edges of G , a weight of $a(b+1)$ on all edges of G' , and a weight 1 on the edge connecting them, sufficiently large integers a and b can make the convergence of HITS arbitrarily slow: in particular, the score of v'_4 eventually grows (arbitrarily) larger than that of v_1 but can be bound below it for an arbitrarily long time. The same can be proved if we remove the $G - G'$ edge and assign weight 1 to the edges of G and $1 + \epsilon$ to the edges of G' ; thus, *if we allow for disconnected graphs, some separation between the weights (e.g. that guaranteed by the integrality constraint) is also necessary*. The (simple) proofs can be found in the full version of the paper [22].

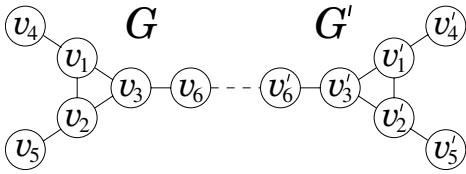


Fig. 1. Pathological edge weights can make HITS converge (arbitrarily) slowly

5 HITS Can Converge Slowly in Rank

The upper bounds of the previous section can be almost matched by:

Theorem 3. *For all $k \geq 3$ and $s \geq 3$ there exists an authority connected graph Γ of maximum degree $2k$ and $3(k + 1)s + k^3 + 2k^2 + 2k + 2 \approx k^3 + 3ks$ vertices on which HITS fails to ϵ -converge on more than $k + 1$ of the top $k^2 + k$ ranks in less than $k^{\Theta(s)}$ iterations for all $\epsilon \leq \bar{\epsilon} = \Theta(\frac{1}{k\sqrt{k}})$.*

Figure 2 provides an example of the graphs on which Theorem 3 proves a glacially slow convergence rate of HITS. Space limitations force us again to leave the proof of the theorem to the final version of the paper [22].

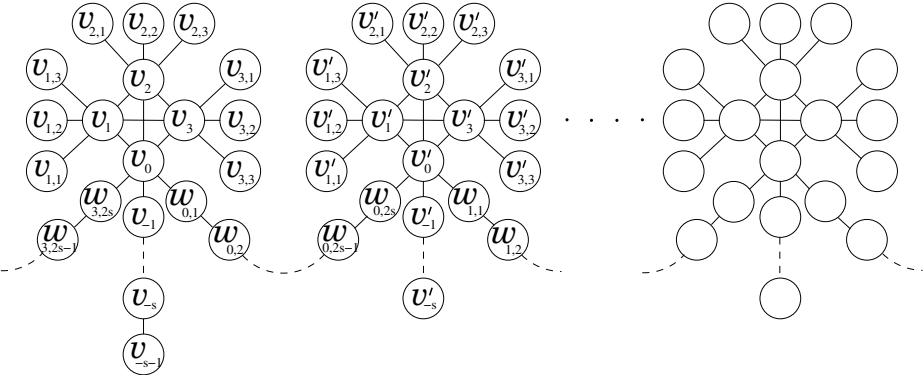


Fig. 2. Γ for $k = 3$. Γ is formed by k “flower” subgraphs $G_1, \dots, G_k$ (right) almost isomorphic to the subgraph G_0 (left, with one more vertex in the stem). Each flower has a corolla of $k + 1$ vertices, all but one with k petals, attached to a stem of $\approx s$ vertices. Strings of $2s$ vertices string the flowers into a circular “garland”.

6 Conclusions and Open Problems

The vast and growing number of applications of HITS (many with little or no connection to Web search) motivates this paper - the first paper to provide non-trivial upper and lower bounds on the convergence of the celebrated algorithm, both in score and, perhaps more importantly, in rank.

We prove that HITS never requires more than $(\lg(\frac{1}{\epsilon}) + \lg(n))(wg)^{\Theta(m^2)}$ iterations to ϵ -converge both in score and on all ranks on any n node graph of maximum degree g whose authority connected components sport at most m nodes and whose links are weighted with integral weights bounded by w . This bound can be tightened somewhat, to $(\lg(\frac{1}{\epsilon}) + \lg(n))(wg)^{\Theta(m)}$ iterations, if the dominant eigenvalues of $\mathbf{A}^T \mathbf{A}$ (where $\mathbf{A}$ is the adjacency matrix of the graph) belong to the same irreducible block - this includes the important class of authority connected graphs. In this case, the integrality condition can also be relaxed into one simply requiring the minimum weight to be 1.

These bounds might appear weak, but acceleration by “repeated squaring” translates them into polynomial upper bounds on the time complexity of reaching up to $\frac{1}{2^{2^{\Theta(n)}}}$ precision, even with $poly(n)$ arc weights: more precisely $O(n^{4+\mu} \lg(n))$ - where $\Theta(n^{2+\mu})$ is the complexity of n by n matrix multiplication - and $O(n^{3+\mu} \lg(n))$ for a large class of graphs, including the important case of authority connected graphs.

Also, we almost match the upper bounds above by exhibiting unweighted authority connected graphs of $\approx k^3 + 3ks$ nodes and maximum degree $2k$ that fail to ϵ -converge on more than $k + 1$ of the top $k^2 + k$ ranks (and thus to ϵ -converge in score) even after $k^{\Theta(s)}$ iterations, for all $\epsilon \leq \bar{\epsilon} = \Theta(\frac{1}{k\sqrt{k}})$ - in other words, HITS fails not only to get the score error below than a not-so-small constant, but fails also to “get right” more than a small fraction of the top ranks unless allowed to run for exponentially many iterations.

Thus, employing repeated squaring acceleration seems absolutely necessary to ensure that one can always reach a satisfactory result in a reasonable time. Graphs of up to a few thousand nodes (like those used in Web search - unlike PageRank, HITS typically operates on small sets of pages preselected through pure textual analysis) seem then certainly tractable. Scaling to beyond a million nodes with convergence guarantees is a challenge probably hard to match even for today’s most powerful computational platforms - unless the application domain ensures the graphs involved meet some “favorable” structural conditions (e.g. in an n node graph with authority connected components of $polylog(n)$ nodes HITS requires complexity $O(n \cdot polylog(n))$ even without sacrificing high accuracy).

Exploring these conditions is certainly a promising direction for future research. It would also be interesting to understand more in depth what conditions one must enforce on link weights to guarantee convergence. Finally, it would be interesting to understand whether the gap in the convergence upper bounds between authority connected and general graphs is indeed fundamental or simply a weakness of our proof techniques.

References

1. Agosti, M., Pretto, L.: A theoretical study of a generalized version of Kleinberg’s HITS algorithm. *Information Retrieval* 8, 219–243 (2005)
2. Bacchin, M., Ferro, N., Melucci, M.: A probabilistic model for stemmer generation. *Information Processing and Management* 41(1), 121–137 (2005)

3. Berry, M. (ed.): *Survey of Text Mining: Clustering, Classification, and Retrieval*. Springer, Heidelberg (2004)
4. Cho, J., Garcia-Molina, H., Page, L.: Efficient crawling through URL ordering. *Computer Networks* 30(1–7), 161–172 (1998)
5. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. In: *Proc. of ACM-SIAM SODA*, pp. 28–36 (2003)
6. Farahat, A., Lofaro, T., Miller, J., Rae, G., Ward, L.: Authority rankings from HITS, PageRank and SALSA: Existence, uniqueness and effect of initialization. *SIAM Journal on Scientific Computing* 27(4), 1181–1201 (2006)
7. Golub, G., Van Loan, C.: *Matrix Computations*, 3rd edn. Johns Hopkins Univ. Press (1996)
8. Haveliwala, T.: Efficient computation of PageRank. Tech. rep., Stanford Un. (1999)
9. Hong, D., Man, S.: Analysis of Web search algorithm HITS. *International Journal of Foundations of Computer Science* 15(4), 649–662 (2004)
10. Jurczyk, P., Agichtein, E.: HITS on question answer portals: Exploration of link analysis for author ranking. In: *Proc. of ACM SIGIR*, pp. 845–846 (2007)
11. Kleinberg, J.: Authoritative sources in a hyperlinked environment. *JACM* 46(5), 604–632 (1999)
12. Knuth, D.: *The Art of Computer Programming*, 3rd edn., vol. 2, sec. 4.6.3. Addison-Wesley, Reading (1998)
13. Kumar, S., Haveliwala, T., Manning, C., Golub, G.: Extrapolation methods for accelerating PageRank computations. In: *Proc. of WWW*, pp. 261–270 (2003)
14. Kurland, O., Lee, L.: PageRank without hyperlinks: Structural re-ranking using links induced by language models. In: *Proc. of ACM SIGIR*, pp. 306–313 (2005)
15. Kurland, O., Lee, L.: Respect my authority! HITS without hyperlinks, utilizing cluster-based language models. In: *Proc. of ACM SIGIR*, pp. 83–90 (2006)
16. Langville, A., Meyer, C.: *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton (2006)
17. Lempel, R., Moran, S.: SALSA: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems* 19(2), 131–160 (2001)
18. Lempel, R., Moran, S.: Rank-stability and rank-similarity of link-based Web ranking algorithms in authority-connected graphs. *Information Retrieval* 8, 219–243 (2005)
19. Mizzaro, S., Robertson, S.: HITS hits TREC - exploring IR evaluation results with network analysis. In: *Proc. of ACM SIGIR*, pp. 479–486 (2007)
20. Peserico, E., Pretto, L.: What does it mean to converge in rank? In: *Proc. of ICTIR*, pp. 239–245 (2007)
21. Peserico, E., Pretto, L.: The rank convergence of HITS can be slow. CoRR, abs/0807.3006 (2008)
22. Peserico, E., Pretto, L.: HITS can converge slowly, but not too slowly, in score and rank, http://www.dei.unipd.it/~pretto/cocoon/hits_convergence.pdf
23. Wang, K., Su, M.-Y.: Item selection by "hub-authority" profit ranking. In: *Proc. of ACM SIGKDD*, pp. 652–657 (2002)

Online Tree Node Assignment with Resource Augmentation

Joseph Wun-Tat Chan¹, Francis Y.L. Chin^{2,*}, Hing-Fung Ting^{2,**},
and Yong Zhang²

¹ Department of Computer Science, King's College London, London, UK
`joseph.chan@kcl.ac.uk`

² Department of Computer Science, The University of Hong Kong, Hong Kong
`{chin,hfting,yzhang}@cs.hku.hk`

Abstract. Given a complete binary tree of height h , the *online tree node assignment problem* is to serve a sequence of assignment/release requests, where an *assignment request*, with an integer parameter $0 \leq i \leq h$, is served by assigning a (tree) node at level (or height) i and a *release request* is served by releasing a specified assigned node. The node assignments have to guarantee that no node is assigned to two assignment requests unreleased, and every leaf-to-root path of the tree contains at most one assigned node. With assigned node reassignments allowed, the target of the problem is to minimize the number of assignments/reassignments, i.e., the cost, to serve the whole sequence of requests. This online tree node assignment problem is fundamental to many applications, including OVSF code assignment in WCDMA networks, buddy memory allocation and hypercube subcube allocation.

Most of the previous results focus on how to achieve good performance when the same amount of resource is given to both the online and the optimal offline algorithms, i.e., one tree. In this paper, we focus on resource augmentation, where the online algorithm is allowed to use more trees than the optimal offline algorithm. By using different approaches, we give (1) a 1-competitive online algorithm, which uses $(h+1)/2$ trees, and is optimal because $(h+1)/2$ trees are required by any online algorithm to match the cost of the optimal offline algorithm with one tree; (2) a 2-competitive algorithm with $3h/8 + 2$ trees; (3) an amortized $(4/3 + \alpha)$ -competitive algorithm with $(11/4 + 4/(3\alpha))$ trees, for any α where $0 < \alpha \leq 4/3$.

1 Introduction

The tree node assignment problem is defined as follows. Given a complete binary tree of height h , the target is to serve a sequence of requests. Every request is classified as either an *assignment request* or a *release request*. To serve an assignment request, which is associated with an integer parameter $0 \leq i \leq h$, we

* Supported by HK RGC grant HKU-7113/07E.

** Supported by HK RGC grant HKU-7171/08E.

have to assign it a (tree) node at level (or height) i . To serve a release request, we just need to mark a specified assigned node free. There are two constraints for the node assignments, which are (1) any node can be assigned to at most one unreleased assignment request, (without ambiguity, all assigned requests are assumed unreleased), and (2) there is at most one assigned node in every leaf-to-root path. Fig. 1 gives a valid tree node assignment.

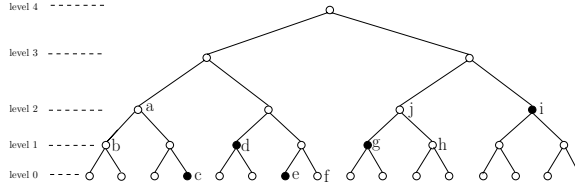


Fig. 1. Example of a valid nodes assignment. Filled circles represent assigned nodes.

The tree node assignment problem can be considered as a general resource allocation problem, which can model the specific problems, such as the Orthogonal Variable Spreading Factor (OVSF) code assignment problem [2, 3, 7, 12, 13, 14, 15, 16, 17], the buddy memory allocation problem [1, 5, 10, 11], and the hypercube subcube allocation problem [6]. The main difference between these problems is how the resource, the nodes at level i , for $0 \leq i \leq h$, are interpreted. In the OVSF code assignment problem, the resource consists of codes of frequency bandwidth 2^i ; in the buddy memory allocation problem, the resource consists of memory blocks of size 2^i ; in the hypercube subcube allocation problem, the resource consists of subcubes of 2^i processors.

Similar to the memory allocation problem, algorithms for the tree node assignment problem also face the *fragmentation problem*. For example, in Fig. 1, there is no node of level 2 that can be assigned (without violating constraint (2) of node assignments). In fact, we can “defragment” the tree by reassigning the assigned node c to free node f . Then, node a is free to assign to assignment request of level 2. In this paper, we consider the tree node assignment problem where *re-assignment* of nodes is allowed. In addition, we design algorithms that serve *all* requests in the request sequence, and we assume that all requests in the request sequence can be served by some algorithm using only one tree of height h .

The performance of an algorithm for the tree node assignment problem is measured by the number assignments/reassignments, which is called the *cost*, carried out by the algorithm. The release operations take no cost, as in the applications, the operation to release a resource is usually negligible when compared with the overhead of assigning/reassigning a resource.

Both the offline and online version of the tree node assignment problem are well studied, especially in the context of OVSF code assignment. In the offline version, the sequence of requests is known in advance, whereas in the online version, the algorithm must process each request without any information about future requests. The offline version of this problem is proved to be NP-hard [16].

Most of the previous work studied the online version of the problem, where performance is given in terms of competitive ratios, i.e., the worst case ratio of the costs between the online algorithm and the optimal offline algorithm. Erlebach et al [7] gave an $O(h)$ -competitive algorithm, where h is the height of the tree, and proved a general lower bound on the competitive ratio of at least 1.5. Forisek et al [8] gave a constant-competitive algorithm, but without deriving the exact value of the constant. Chin, Ting and Zhang [2] gave a 10-competitive algorithm and, in addition, their algorithm guarantees that each request is served with at most 5 assignments/reassignments. They then improved the upper bound by proposing a 6-competitive algorithm [3]. They improved the lower bound of the competitive ratio to $5/3 \approx 1.67$ [2]. Very recently, Miyazaki and Okamoto [14] gave a 7-competitive algorithm and improved the lower bound of the competitive ratio to 2.

In this paper, we focus on the online version of the problem with resource augmentation [9], which means that the online algorithm is allowed to use more trees than the optimal offline algorithm. We assume that the optimal offline algorithm uses one tree, while the online algorithm can use k trees, where $k \geq 1$. The competitive ratio is defined to be the worst case ratio of the cost between the online algorithm with k trees and the optimal offline algorithm with one tree. This problem has been studied before. Erlebach et al [7] gave a 4-competitive algorithm with two trees, and Chin, Zhang and Zhu [4] gave a 5-competitive algorithm with $9/8$ trees.

The main contribution of this paper is to show how the competitive ratio can be further reduced by making use of more trees. In other words, how the future information (offline) can be compensated by extra resources (trees). First, we give an online algorithm with $(h+1)/2$ trees that matches the cost of the optimal offline algorithm with one tree. In fact, this algorithm even matches the cost of each request with that of the optimal offline algorithm with one tree, as it does not require any reassignments. We further show that for any online algorithm to match the cost of the optimal offline algorithm with one tree, $(h+1)/2$ trees are necessary. That implies that our 1-competitive algorithm is optimal in terms of the number of trees used. Then, by using one extra reassignment for each release request to reduce the fragmentation of the assigned nodes in the tree, we can use fewer trees to serve the sequence of requests. In particular, we give a 2-competitive online algorithm with $3h/8 + 2$ trees. This algorithm bounds the cost of each request to one, i.e., each assignment request takes one assignment and each release request takes at most one reassignment. These two algorithms with bounded costs for each request are presented in Section 2.

When it is not necessary to bound the cost of individual requests to a constant, we can achieve an amortized $(4/3+\alpha)$ -competitive algorithm with $(11/4+4/(3\alpha))$ trees, for any α where $0 < \alpha \leq 4/3$. This algorithm is presented in Section 3.

Remark: Because of page limit, some detailed proofs are removed. For full version of this paper, please refer to <http://www.cs.hku.hk/~yzhang/tree.pdf>

2 Algorithms with Bounded Cost per Request

We present two algorithms in this section. For any request sequence σ where the optimal algorithm can satisfy the requests with one tree,

- the first algorithm uses at most $(h + 1)/2$ trees and incurs a cost of one for each assignment request and a cost of zero for each release request;
- the second algorithm uses at most $3h/8 + 2$ trees and incurs a cost of at most one for each assignment and release request.

Since any algorithm needs to assign a node for each assignment request, the first algorithm is optimal in terms of the cost, i.e., 1-competitive. We further show that it is necessary for any online algorithm to use $(h + 1)/2$ trees in order to match the cost of the optimal algorithm with one tree. Since the number of release requests is at most the number of assignment requests, the total cost incurred by the second algorithm is at most twice that of the optimal algorithm with one tree, i.e., 2-competitive.

2.1 Preliminary

We introduce some definitions in this part, which are used in subsequent sections. A node v is called a *free node* if there is no assigned node in any leaf-to-root path going through the node v . A node v is *blocked* if there is a path from the root to a leaf through the node v that contains an assigned node. Node v is also called a *blocked node*, moreover, we say node v is *blocked by an assigned node at some level*. A node at level i or an assignment request that asks for a node at level i is said to have a *bandwidth* of 2^i . It is clear that to serve a set of assignment requests or to accommodate a set of assigned nodes in a tree of height h , the total bandwidth of the requests or nodes has to be no more than 2^h .

2.2 Optimal 1-Competitive Algorithm

The idea to achieve the optimal cost is to dedicate some subtrees to serve assignment requests of some particular levels. To describe this assignment scheme, we define a *half-tree* to be the subtree rooted at either the left or right child of a root. This online algorithm uses $h + 1$ half-trees. We label the $h + 1$ half-trees from 0 to h . When there is a level- i assignment request with $i < h$, we pick from half-trees 0 to $i + 1$ any free node at level i and assign it to the request. If $i = h$, we assign the root of any tree to the request, as there should be no other assigned nodes. For any release request, we just release the assigned node and mark it free.

The correctness of the algorithm depends on whether, for each level- i assignment request, we can always find a level- i free node from the half-trees 0 to $i + 1$. The following lemma makes sure that it can be done.

Lemma 1. *If the total bandwidth of the assigned nodes and the new-coming assignment request of level i is less than 2^h , there is always a free node at level i in the half-trees 0 to $i + 1$ for $i < h$.*

Theorem 2. *For the online tree node assignment problem, we have an 1-competitive algorithm using $(h + 1)/2$ trees, where the cost of serving each assignment request is one and the cost of serving each release request is zero.*

Lower bound of number of trees to achieve 1-competitiveness. We give an adversary such that the optimal algorithm with one tree serves each assignment request with only one assignment and each release request with no reassignment. At the same time, for any online algorithm that wants to limit the cost as the optimal algorithm, the adversary forces it to use at least $(h + 1)/2$ trees.

The main idea of the adversary is to send assignment requests in ascending order of their levels. The adversary then releases some requests but makes sure that the remaining assigned nodes of low level block a significant part of the trees. Thus, the assignment requests of high level need to be served with extra trees. The adversary is divided into h steps, where in Step i , assignment requests of level i are sent, and then some release requests of level $j \leq i$ follow, except for Step $h - 1$. Over all time, the total bandwidth of the assigned nodes is kept at most 2^h . The details of the adversary is given as follows.

Step 0: The adversary sends 2^h level-0 assignment requests. Any online algorithm must assign 2^h level-0 free nodes. The adversary then releases 2^{h-1} of the level-0 assigned nodes such that $2^{h-1} = 2 \cdot 2^{h-2}$ level-1 nodes are blocked.

Step 1: The adversary sends 2^{h-2} level-1 assignment requests. After the online algorithm has assigned 2^{h-2} level-1 free nodes, the adversary releases 2^{h-2} level-0 and 2^{h-3} level-1 assigned nodes, i.e., half of the assigned nodes at each level with assigned nodes. The release requests make sure that it results in $3 \cdot 2^{h-3}$ nodes blocked at level-2.

...

Step i , for $2 \leq i \leq h - 2$: The adversary sends 2^{h-i-1} level- i assignment requests. After the online algorithm has assigned 2^{h-i-1} level- i free nodes, the adversary releases 2^{h-i-1} level-0 assigned nodes and 2^{h-i-2} level- j assigned nodes for $1 \leq j \leq i$, i.e., half of the assigned nodes at each level with assigned nodes. The release requests make sure that it results in $(i + 2) \cdot 2^{h-i-2}$ nodes blocked at level- $(i + 1)$.

...

Step $h - 1$: The adversary sends one level- $(h - 1)$ assignment requests. (Now, there are $h + 1$ nodes blocked at level $h - 1$.)

Lemma 3. *For any online algorithm, at the end of Step i , the number of level- $(i + 1)$ nodes blocked is $(i + 2) \cdot 2^{h-i-2}$, for $0 \leq i \leq h - 2$.*

It is easy to construct an offline algorithm with one tree so that it serves for the adversary each assignment request with one assignment and each release request with no reassignment. Thus, we have the following theorem.

Theorem 4. *No online algorithm can match the cost of the optimal offline algorithm with one tree by using less than $(h + 1)/2$ trees.*

2.3 2-Competitive Algorithm with Bound Cost per Request

This section gives an online algorithm that uses fewer trees, i.e., $3h/8 + 2$, but comes with a slightly higher competitive ratio, i.e., 2. The main idea of the algorithm is to apply an extra reassignment for any release request to reduce fragmentation of the tree (to reduce blocking bandwidth) by pairing two assigned nodes with unassigned siblings. Precisely, the algorithm serves each assignment and release request with at most one assignment or reassignment. As the number of release requests is at most the number of assignment requests, the total cost of the algorithm is at most twice that of the optimal algorithm.

We design an assignment scheme for the $3h/8 + 2$ trees available to the online algorithm. First, we define an *eighth-tree* to be a subtree rooted at a level- $(h-3)$ node. All the $3h$ eighth-trees in the $3h/8$ trees are labeled. Six of them are labeled 0 and three of them are labeled i , for $1 \leq i \leq h-2$. Denote the other two trees as T and T^* . In general, the $3h$ eighth-trees are to handle the assignment requests at level- i for $0 \leq i \leq h-3$, T for level- $(h-2)$ and $-(h-1)$, and T^* for all levels. In particular, at any time, we allow at most one assigned node in each level $0 \leq i \leq h$ of T^* . It enables us to find a free node at level- i of T^* , whenever there is no assigned node at level- i .

The details of the assignment scheme are given as follows.

Assignment request R of level i :

If there is no level- i assigned node in T^* , assign R a level- i free node in T^* . Otherwise,

- If $i = h-2$ or $i = h-1$, assign R any level- i free node in T .
- If $0 \leq i \leq h-3$, assign R any level- i free node from any eighth-tree labeled k for $0 \leq k \leq i$. If no free node is available, consider the eighth-trees labeled $i+1$. If there is a level- i free node v where v 's sibling is an assigned (level- i) node, assign R the free node v . Otherwise, assign R any level- i free node in any eighth-tree labeled $i+1$. (Lemma 4 shows that an level- i free node always exists in one of the eighth-tree with label from 0 to $i+1$).

Release request R of level i :

Release the node assigned to R and mark it free.

If $0 \leq i \leq h-3$, consider the following situations for reassignment.

- If there is no assigned node at level i of T^* and there is a level- i assigned node v in an eighth-tree labeled $i+1$ where v 's sibling is a free (level- i) node, reassign v to a level- i free node of T^* .
- If there are two level- i assigned node u and v in an eighth-trees labeled $i+1$ where both u 's and v 's siblings are a free (level- i) node, reassign u to v 's sibling.

To ensure the correctness of the algorithm, we show that the followings are true.

1. When T^* contains an assigned node at level i , for $i = h-1$ or $i = h-2$, there is always a level- i free node in T . For this case, it is clear as otherwise, the total bandwidth of the assigned nodes is at least 2^h .

2. When T^* contains an assigned node at some level i ($0 \leq i \leq h-3$), there is always a level- i free node in some eighth-tree labeled k , for $0 \leq k \leq i+1$.

In order to ensure that the Property (2) is true (by Lemma 6), we may spend an extra reassignment after a release request to tidy up the configuration of the assigned nodes in the eighth-trees. We want to maintain a configuration of the assigned nodes in the eighth-trees as in the following lemma.

Lemma 5. *Let $0 \leq i \leq h-3$. (1) When T^* contains no assigned node at level i , there is no assigned node v at level i of any eight-tree labeled $i+1$ where v 's sibling is a free node. (2) If there exist one assigned node v , among all assigned nodes, at level i of some eight-tree labeled $i+1$ where v 's sibling is a free node, T^* must contains an assigned node at level i .*

Lemma 6. *Assume that the total bandwidth of the assigned nodes is less than 2^h . For any i , $0 \leq i \leq h-3$, when T^* contains an assigned node at level i , there is always a level- i free node in some eighth-tree labeled k , for $0 \leq k \leq i+1$.*

Theorem 7. *For the online tree node assignment problem, we have a 2-competitive algorithm using $3h/8 + 2$ trees and the cost of serving each request is at most one.*

3 Algorithm with Amortized Constant Cost per Request

In this section, we give a $(4/3 + \alpha)$ -competitive algorithm using $11/4 + (4/(3\alpha))$ trees, for any α where $0 < \alpha \leq 4/3$. This algorithm is based on an extended concept of *compact configuration* of assigned nodes in trees [7], which is described below.

Assume that the available trees to the online algorithm are arranged on a line. For any two nodes u and v , where u is not an ancestor of v and vice versa, we say that u is on the *left* of v if u is in a tree which is on the left of the tree containing v , or there is a leaf in the subtree rooted at u which is on the left of a leaf in the subtree rooted at v ; otherwise, u is on the *right* of v . Level i of the trees is defined to be *compact* if all nodes of level i to the left of a blocked node of level i , which is blocked by an assigned node at level no more than i , are also blocked. We say that a configuration is compact if all levels of the trees are compact.

It is very costly to maintain a compact configuration after serving each request. By using a “relaxed compact” configuration with a constant number of trees, the amortized competitive ratio can be reduced to a constant. The idea of the relaxation is as follows: for each level i , there may exist more free nodes than the compact configuration. Such kind of free nodes can accommodate the following assignment request immediately without reassigning nodes at higher levels. Thus, there may be no extra cost (reassignment) after serving some request and the amortization of each request can be reduced to a constant by using some extra resources (the free node to the right of some assigned nodes may be wasted).

To improve the competitive ratio, we could be lazy in tidying up the configuration when serving assignment or release requests. In this part, we define a less “tidy” configuration called the *semi-compact configuration*, which stores odd-level and even-level assigned nodes separately. However, the way to store the two sets of assigned nodes is the same. For simplicity, we would show how even-level assigned nodes are stored only.

To describe the algorithm, we define a notation called the *level- i region*, which consists of all level- i assigned nodes and maybe some level- i free nodes. There is a level- i region for each level i . If there is no assigned nodes at level i , the level- i region may consist of only free level- i nodes or an empty region.

The semi-compact configuration (as shown in Figure 2) divides the level- i region into two contiguous parts, the *main region* on the left and the *gap region* on the right.

- The main region consists of assigned nodes and maybe some free nodes, but the number of free nodes is at most β times the number of assigned nodes, where $\beta \geq 1$ is a fixed parameter.
- The gap region consists of only free nodes and the number of free nodes is at most 7.

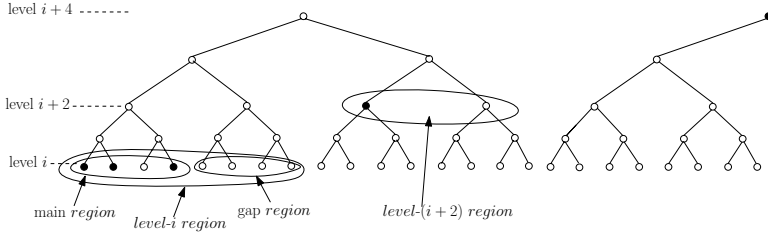


Fig. 2. An example of an semi-compact configuration

The details of the algorithm is given as follows.

Assignment request R of level i :

- Case A1.** If there is a level- i free node in the level- i main region, assign R the free node.
- Case A2.** If there is a level- i free node in the level- i gap region, assign R the leftmost free node, say u , and u is moved from the gap to the main region.
- Case A3.** If there is no level- i free node in the level- i region, find the non-empty level- j region G_j for the smallest $j > i$
- (1) If G_j has no assigned node, i.e., the leftmost node is a free node, the free node is “divided” into four level- i free nodes, three level- k nodes for even-level k between $i+2$ to $j-2$. These free nodes are inserted to the corresponding level- i and level- k gap regions. The leftmost level- i free node is assigned to R . Similar to the case A2, the newly assigned node is moved from the gap to the main region.

- (2) Otherwise, release the leftmost assigned node, say u , of G_j , which is reassigned later. The released node is divided and assigned as in step (1). An assignment request of level j is issued to find a free node for u .

Release request R of level i :

Release the assigned node for R .

Case R1. If the number of free nodes in the level- i main region is at most β times the number of assigned nodes, do nothing.

Case R2. If the number of free nodes in the level- i main region is more than β times the number of assigned nodes, compact the main region into contiguous assigned nodes on the left by reassignments. The free nodes are moved to the gap regions.

- (3) If the number of free nodes in the gap region is at most 7, do nothing.
- (4) If the number of free nodes in the gap region is more than 7, let the number be in the form $4x + y$ where x and y are integers and $x > 1$ and $4 \leq y \leq 7$. Group the rightmost $4x$ free nodes into x level- $(i + 2)$ free nodes. The x level- $(i + 2)$ free nodes are moved to the level- $(i + 2)$ main region in a way that are considered as x release requests.

We use an amortized analysis to bound the average number of assignments and reassignments needed for serving each assignment or release request. The credits paid for an assignment request is $4/3$ and a release request is α where $\alpha = 4/(3\beta) \leq 4/3$ as $\beta \geq 1$. The potential of a level- i main region is α times the number of free nodes in the main region. The potential of the level- i gap region is defined as follows.

Number of free nodes in the gap region	0	1	2	3	4	5	6	7
Potential	1	2/3	1/3	0	0	$\alpha/4$	$\alpha/2$	$3\alpha/4$

The potential of a semi-compact configuration is the sum of all potential of the level- i main and gap regions for all level i . The initial semi-compact configuration has four level-0 free nodes in the level-0 gap region, and three level- i free nodes in the level- i gap region for all other $i > 0$. All main regions are empty. The initial potential is 0.

The following lemma shows that the saved credit is able to pay the actual cost of the algorithm for serving each request.

Lemma 8. *Let S_b and S_a be the potential of the semi-compact configuration before and after serving a request. The actual cost to serve an assignment request is at most $4/3 - (S_a - S_b)$ and a release request is at most $\alpha - (S_a - S_b)$.*

Summing up all node trees used, we can prove the following theorem.

Theorem 9. *For the online tree node assignment problem, our algorithm in this section is $(4/3 + \alpha)$ -competitive and it uses at most $11/4 + 4/(3\alpha)$ trees, for any α where $0 < \alpha \leq 4/3$.*

References

1. Brodal, G.S., Demaine, E.D., Munro, J.I.: Fast allocation and deallocation with an improved buddy system. *Acta Inf.* 41(4-5), 273–291 (2005)
2. Chin, F.Y.L., Ting, H.F., Zhang, Y.: A constant-competitive algorithm for online OVFS code assignment. In: Tokuyama, T. (ed.) *ISAAC 2007*. LNCS, vol. 4835, pp. 452–463. Springer, Heidelberg (2007)
3. Chin, F.Y.L., Ting, H.F., Zhang, Y.: Constant-Competitive Tree Node Assignment (manuscript)
4. Chin, F.Y.L., Zhang, Y., Zhu, H.: Online OVFS code assignment with resource augmentation. In: Kao, M.-Y., Li, X.-Y. (eds.) *AAIM 2007*. LNCS, vol. 4508, pp. 191–200. Springer, Heidelberg (2007)
5. Defoe, D.C., Cholleti, S.R., Cytron, R.: Upper bound for defragmenting buddy heaps. In: *Proceedings of the 2005 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pp. 222–229 (2005)
6. Dutt, S., Hayes, J.P.: Subcube allocation in hypercube computers. *IEEE Trans. Computers* 40(3), 341–352 (1991)
7. Erlebach, T., Jacob, R., Mihalák, M., Nunkesser, M., Szabó, G., Widmayer, P.: An algorithmic view on OVFS code assignment. *Algorithmica* 47(3), 269–298 (2007)
8. Forišek, M., Katreniak, B., Katreniaková, J., Královič, R., Královič, R., Koutný, V., Pardubská, D., Plachetka, T., Rován, B.: Online bandwidth allocation. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) *ESA 2007*. LNCS, vol. 4698, pp. 546–557. Springer, Heidelberg (2007)
9. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. *J. ACM* 47(4), 617–643 (2000)
10. Knowlton, K.C.: A fast storage allocator. *Commun. ACM* 8(10), 623–624 (1965)
11. Knuth, D.E.: *The Art of Computer Programming. Fundamental Algorithms*, vol. 1. Addison-Wesley, Reading (1975)
12. Li, X.-Y., Wan, P.-J.: Theoretically good distributed CDMA/OVFS code assignment for wireless ad hoc networks. In: Wang, L. (ed.) *COCOON 2005*. LNCS, vol. 3595, pp. 126–135. Springer, Heidelberg (2005)
13. Minn, T., Siu, K.-Y.: Dynamic assignment of orthogonal variable-spreading-factor codes in W-CDMA. *IEEE Journal on Selected Areas in Communications* 18(8), 1429–1440 (2000)
14. Miyazaki, S., Okamoto, K.: Improving the competitive ratio of the online OVFS code assignment problem. In: *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC)*, pp. 64–76 (2008)
15. Rouskas, A.N., Skoutas, D.N.: OVFS codes assignment and reassignment at the forward link of W-CDMA 3G systems. In: *Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, vol. 5, pp. 2404–2408 (2002)
16. Erlebach, T., Jacob, R., Tomamichel, M.: Algorithmische Aspekte von OVFS Code Assignment mit Schwerpunkt auf Offline Code Assignment. Student thesis at ETH Zürich
17. Wan, P.-J., Li, X.-Y., Frieder, O.: OVFS-CDMA code assignment in wireless ad hoc networks. *Algorithmica* 49(4), 264–285 (2007)

Why Locally-Fair Maximal Flows in Client-Server Networks Perform Well

Kenneth A. Berman and Chad Yoshikawa*

University of Cincinnati, Computer Science Department,
Cincinnati, Ohio 45221, United States of America
{ken.berman,chad.yoshikawa}@uc.edu
<http://www.cs.uc.edu/>

Abstract. Maximal flows reach at least a $1/2$ approximation of the maximum flow in client-server networks. By adding only 1 additional time round to any distributed maximal flow algorithm we show how this $1/2$ -approximation can be improved on bounded-degree networks. We call these modified maximal flows ‘locally fair’ since there is a measure of fairness prescribed to each client and server in the network. Let $N = (U, V, E, b)$ represent a client-server network with clients U , servers V , network links E , and node capacities b , where we assume that each capacity is at least one unit. Let $d(u)$ denote the b -weighted degree of any node $u \in U \cup V$, $\Delta = \max\{d(u) | u \in U\}$ and $\delta = \min\{d(v) | v \in V\}$. We show that a locally-fair maximal flow f achieves an approximation to the maximum flow of $\min\{1, \frac{\Delta^2 - \delta}{2\Delta^2 - \delta\Delta - \Delta}\}$, and this result is sharp for any given integers δ and Δ . This results are of practical importance since local-fairness loosely models the steady-state behavior of TCP/IP and these types of degree-bounds often occur naturally (or are easy to enforce) in real client-server systems.

Keywords: Distributed Flow Algorithms, Oblivious Routing, Network Algorithms, Maximal Flow.

1 Introduction

Using a maximal flow to approximate the maximum flow is attractive for client-server networks since (i) it can be computed without global information, (ii) it can be computed quickly, and (iii) it reaches a $1/2$ -approximation of the maximum throughput. For example, maximal flows are the basis for the ‘Restricted Adversary’ routing algorithm of Awerbuch, Hajiaghayi, Kleinberg, and Leighton [2] and the ‘Aggressive Increase’ routing algorithm of Annexstein, Berman, Strunjas, and Yoshikawa in [1] – both of which are locally-adaptive algorithms that reach constant-competitiveness in $O(\log(\Delta))$ distributed time rounds where Δ is the maximum client degree in the network. Other examples of distributed maximal flow algorithms include the ‘BipartiteMatch’ and

* Supported in part by NSF Grant 0521189 and an Ohio Board of Regents Ph.D. Fellowship.

‘LowDegreeMatch’ distributed algorithms of Hańćkowiak, Karoński, and Pancónesi [6] which compute a maximal flow on n -node client-server networks in $\text{polylog}(n)$ and $O(\Delta)$ time rounds, respectively.

In general, a maximal flow on an arbitrary client-server network cannot do better than a $1/2$ -approximation of the maximum flow. However, maximal flows generally outperform this approximation in practice. For example, in [1] it was shown that the aforementioned ‘Restricted Adversary’ and ‘Aggressive Increase’ distributed maximal-flow algorithms reach better than 75% and 90%, respectively, of the optimal throughput on a variety of bipartite graphs that are described in [4]. On the other hand, there exist extremal graphs for which these algorithms can only attain the $1/2$ lower bound of maximal flows. The question is, under what set of conditions does a maximal flow perform well?

In this paper, we show that we can do much better than a $1/2$ -approximation if the network satisfies certain degree conditions and the maximal flow f is *locally-fair*. More precisely, let $N = (G = (U \cup V, E), b)$ be a client-server network, where G is a bipartite graph with node set $U \cup V$ and edge set E representing the topology of the network, i.e., $uv \in E$ if and only if a communication link has been established between client u and server v , and b is a capacity weighting of the nodes $U \cup V$. (For a client u , $b(u)$ represents the demand of client u .) A flow f is a weighting of E with nonnegative real numbers such that, for each $x \in U \cup V$, $f(x) \leq b(x)$, where $f(x) = \sum_{xy \in E} f(xy)$. The *size* of a flow f , which we denote by $\text{size}(f)$, is the total flow over all the edges, i.e., $\text{size}(f) = \sum_{e \in E} f(e)$. A maximum flow is one that maximizes $\text{size}(f)$. A maximal flow is one such that for each $uv \in E$, either client u or server v is saturated, where a node $x \in U \cup V$ is *saturated* with respect to f , if $f(x) = b(x)$.

1.1 Locally-Fair Flows

First consider the case where each client sends at most one unit of flow, i.e., $b(u) = 1, u \in U$. In this case, if a server were to reserve an equal portion of its capacity for each neighborhood client u , then u would receive its “fair” share of the server’s capacity equal to $b(v)/d(v)$, where $d(v)$ is the degree of v . We consider flows f that are “client locally fair” in the sense that, either client u is saturated (is fully satisfied), or u sends a flow to each neighborhood server v , which is at least as great as its fair share of v ’s capacity (i.e., at least $b(v)/d(v)$). This definition extends to the general case, where clients may have non-unit capacities, except that fair share becomes $b(u)b(v)/d(v)$, where $d(v)$ becomes the b -weighted degree $d_b(v)$. For $x \in U \cup V$, the b -weighted degree of x is the total capacity over all nodes in the neighborhood of x , i.e., $d_b(x) = \sum_{xy \in E} b(y)$. For convenience, in the remainder of this paper we will denote $d_b(x)$ simply by $d(x)$.

In this paper we consider flows, which we call “locally-fair”, that are not only locally fair to the clients as described above, but satisfy the additional condition that, if a client u is saturated and does not send its fair share of v ’s capacity to server v , then v receives at least its fair share of u ’s demand (i.e., at least $b(u)b(v)/d(u)$).

Definition 1. A flow f is locally-fair if, for each edge $uv \in E$, $f(uv) \geq \min \left\{ \frac{b(u)b(v)}{d(u)}, \frac{b(u)b(v)}{d(v)} \right\}$ and if the client u is unsaturated, $f(uv) \geq \frac{b(u)b(v)}{d(v)}$.

The following proposition shows that locally-fair flows always exist.

Proposition 1. Let $N = (G = (U \cup V, E), b)$ be a client-server network. Then, there always exists a locally-fair flow f .

Proof. Let μ be the edge-weighting given by $\mu(uv) = \min(\frac{b(u)b(v)}{d(u)}, \frac{b(u)b(v)}{d(v)})$. Clearly, μ satisfies the capacity constraints on the nodes, i.e., is a flow in the network. For any client that is not saturated, i.e. a client u for which $\sum_{uv \in E} \mu(uv) < b(u)$, increase the weight on each of its outgoing edges uv up to $b(u)b(v)/d(v)$ (or less) until the client is saturated or until all edges have been exhausted. The resulting flow f is locally-fair.

Based on the proof of Proposition 1 a locally-fair flow can be computed in an distributed setting using only a few communication steps.

Locally-Fair Flow (LFF) Algorithm

Input: Distributed client-server network $N = (G = (U \cup V, E), b)$

Output: Distributed locally-fair flow f in N

1. **for each** server $v \in V$ **do in parallel**
 server v sends the values $b(v)$ and $d(v)$ to each client u in
 its neighborhood
 2. **for each** client $u \in U$ **do in parallel**
 client u assigns initial flow values $\min\{b(u)b(v)/d(u), b(u)b(v)/d(v)\}$
 for each edge uv
 while client u not saturated
 client increases flow values on each edge uv up to the
 value $b(u)b(v)/d(v)$
-

Note that any extension of f is necessarily locally-fair. Using more rounds, the initial flow generated by algorithm *LFF* can be extended to a locally-fair *maximal* flow by computing a maximal flow g using any distributed maximal flow algorithm (see [1,6] for references to such algorithms) with respect to the residual capacities (where the residual capacity of node $x \in U \cup V$ is $b(x) - f(x)$) and increasing the flow on edge e by $g(e)$. On an n -node client-server network, these distributed maximal flow algorithms are $O(\Delta)$ and $O(\text{polylog}(n))$, respectively. We can also obtain a locally-fair maximal flow by repeatedly calling *LFF* with residual capacities.

Locally-Fair Maximal Flow (LFMF) Algorithm

Input: Distributed client-server network $N = (G = (U \cup V, E), b)$

Output: Distributed locally-fair maximal flow f in N

1. **call** LFF to obtain a distributed flow f
 2. **while** there exist an edge uv where both u and v are unsaturated
 - call** LFF with capacity $b(x)$ replaced with residual capacity $r(x) = b(x) - f(x)$,
 $x \in U \cup V$, to obtain a distributed flow f_r and add this flow to f
-

After the first round each client u sends a flow of $f(uv)$ to each neighborhood server v and increases this flow by $f_r(uv)$ after each subsequent round. Algorithm $LFMF$ is actually a special case of an aggressive increase algorithm as described in [1], under the slightly more general condition that clients may have non-unit capacities, and with the additional demand that $f(uv) \geq \mu(uv)$ for all $uv \in E$. Algorithm $LFMF$ performs at most Δ_V rounds where Δ_V denotes the maximum (un-weighted) degree of a server. To see this observe that the flow f output by LFF has the property that each server s is either saturated, or at least one neighborhood client of s is saturated. This follows from the fact that unsaturated clients send their fair-share to s and therefore s would be saturated by f if all neighborhood clients were unsaturated.

The following proposition shows that Algorithm $LFMF$ is very close to maximal after $O(\log \Delta_U)$ rounds, where Δ_U denotes the maximum (un-weighted) degree of a client.

Proposition 2. *After performing $2 \log_2(\Delta_U/\epsilon) + 4$ rounds the flow f generated by algorithm $LFMF$ is a $\frac{1}{1+\epsilon}$ -approximation to a locally-fair maximal flow, i.e., $size(f) = (\frac{1}{1+\epsilon})size(g)$ for some locally-fair maximal flow g .*

The proof is given in the full version of the paper.

In this paper we consider the following natural degree constraints, which are practical in a distributed setting and enforceable without global communication, for which algorithm $LFMF$ achieves much better than a $1/2$ -approximation.

(Δ, δ) Dual-Bounded Network. A network such that every client has b -weighted degree at most Δ and every server has b -weighted degree at least δ .

Our main result is the following theorem.

Theorem 1. *Let $N = (G = (U \cup V, E), b)$ be a (Δ, δ) -dual-bounded network, where $b(x) \geq 1$ for all $x \in U \cup V$, and let f be any locally-fair maximal flow. If $\delta \geq \Delta$ then f is a maximum flow; otherwise, f achieves an approximation to a maximum flow of at least $\frac{\Delta^2 - \delta}{2\Delta^2 - \delta\Delta - \Delta}$. Further, this result is sharp for any given integers δ and Δ .*

Given a fraction p , it immediately follows from Theorem 1 that a p -approximation is achieved if

$$\delta \geq \frac{\Delta^2(2p-1) - \Delta p}{\Delta p - 1}.$$

For example, suppose $\Delta = 10$. Then a .9-approximation is achieved if $\delta \geq 8.875$.

2 Related Work on Tight Bounds for Maximal Flows

As far as we know, Theorem 1 is the first tight bound on maximal flows besides the aforementioned $1/2$ -bound. There is work done by Cardinal, Labbé, Langerman, Levy, and Mélot [3] which is similar in spirit in that a bound is given for the approximation that a maximal flow yields for the vertex covering problem.

Similar to our work is that of Czygrinow, M. Hańćkowiak and Szymańska [5] which presented an algorithm for extending a maximal matching to achieve a $2/3$ -approximation to the maximum matching. In this work, the authors used the method of removing short augmenting paths in order to improve the matching size. However, instead of requiring 1 additional round of distributed computation the algorithm in that work required an additional $\text{polylog}(n)$ additional time rounds where n is the number of nodes in the graph.

3 Dual Bounds

In this section, we prove Theorem 1 which relates the size of a locally-fair flow to the maximum flow in terms of the dual-bounds Δ and δ .

First, we provide the following lemma which guarantees that every server $v \in N$ receives at least $b(v) \min(1, \delta/\Delta)$ flow.

Lemma 1. *In a locally-fair flow every server $v \in V$ receives flow at least $\min\{\frac{\delta}{\Delta}b(v), b(v)\}$.*

Proof. Pick any server v . If $d(v) \geq \Delta$ then each incoming edge uv carries flow of $b(u)b(v)/d(v)$ and the total incoming flow to v is $b(v)$ and the server is saturated. Otherwise, each incoming edge uv carries flow at least $b(u)b(v)/\Delta$ which sum to at least $b(v)\min(1, \delta/\Delta)$.

Corollary 1. *In a network N , if $\delta \geq \Delta$, then a locally-fair flow is maximum and has size equal to the total capacity of the servers, s .*

The corollary follows immediately from Lemma 1.

Now we will prove Theorem 1 which is repeated here:

Claim. Let $N = (G = (U \cup V, E), b)$ be a (Δ, δ) -dual-bounded network, where $b(x) \geq 1$ for all $x \in U \cup V$, and let f be any locally-fair maximal flow. If $\delta \geq \Delta$ then f is a maximum flow; otherwise, f achieves an approximation to the maximum flow of at least $\frac{\Delta^2 - \delta}{2\Delta^2 - \delta\Delta - \Delta}$. Further, this result is sharp for any given integers δ and Δ .

Proof. Let f be any locally-fair maximal flow having size $\phi = \sum_{e \in E} f(e)$. If $\delta \geq \Delta$, then by Corollary 1, f is a maximum flow. Therefore, we can assume that $\delta < \Delta$ in the rest of the proof.

We will prove the theorem by lower-bounding the flow of first the servers and then the clients. First, the flow of the servers is examined. For convenience, we partition the clients U and servers V into the following sets:

- W the unsaturated clients
- X the saturated clients (also the client-complement of W)
- Y the neighborhood of W , i.e. $N(W)$
- Z the server-complement of Y

For any set S , we denote as f_S the size of the flow entering (or leaving) the set S , i.e., $f_S = \sum_{x \in S, y \in E} f(xy)$. For convenience, hereafter we refer to flow size as “flow” whenever the usage is unambiguous. Using this notation, the flow entering the servers is given by:

$$\phi = f_V = f_Y + f_Z$$

Note that all servers in $Y = N(W)$ are necessarily saturated since (1) clients in W are unsaturated by definition and (2) any maximal flow induces a vertex cover of saturated clients and saturated servers. Thus, the flow size can be rewritten as:

$$f_V = b(Y) + f_Z$$

where $b(Y)$ is the total capacity of the servers in set Y , i.e. $b(Y) = \sum_{y \in Y} b(y)$.

Using Lemma 1, every server $z \in Z$ must receive flow at least $\frac{\delta}{\Delta}b(z)$. Summing over all $z \in Z$, then, the flow size f_Z is at least $b(Z)\frac{\delta}{\Delta}$. Thus, the flow size into the servers, f_V , satisfies the following inequality:

$$f_V \geq b(Y) + b(Z)\frac{\delta}{\Delta} \quad (1)$$

Now, we examine the flow leaving the client set U . The flow leaving the clients, f_U , is given by:

$$\phi = f_U = f_X + f_W = b(X) + f_W$$

since all nodes in X are saturated by definition.

The flow f_W is equivalent to the flow entering the server set Y restricted to the clients in W . By definition of a locally-fair flow, since every client in W is unsaturated then each client $w \in W$ will always send flow at least $\frac{b(w)b(y)}{d(y)}$ for each $y \in N(w)$.

First, let us define the notation that $d_W(y)$ is the b -weighted degree of node y restricted to the set W , e.g. $d_W(y) = \sum_{w \in N(y) \cap W} b(w)$. Using this notation we have the following inequality:

$$f_W \geq \sum_{y \in Y} b(y) \frac{d_W(y)}{d(y)}$$

$$\begin{aligned}
&= \sum_{y \in Y} b(y) \frac{d_W(y)}{d_W(y) + d_X(y)} \\
&\geq \sum_{y \in Y} b(y) \frac{1}{1 + d_X(y)}
\end{aligned}$$

where the last inequality holds since all b -values are at least 1 and Y is the neighborhood of W .

Thus, we can rewrite the bound on total client flow, f_U , as:

$$\begin{aligned}
f_U &\geq b(X) + \sum_{y \in Y} b(y) \frac{1}{1 + d_X(y)} \\
&\geq b(X) + \frac{b(Y)^2}{b(Y) + \sum_{y \in Y} b(y)d_X(y)}
\end{aligned}$$

where the second line follows from the Arithmetic Mean-Harmonic Mean (AM-HM) inequality.

The sum $\sum_{y \in Y} b(y)d_X(y)$ can be bounded using the definitions of Δ and δ :

$$\begin{aligned}
\sum_{x \in X} b(x)\Delta &\geq \sum_{x \in X} (b(x)d_Y(x) + b(x)d_Z(x)) \\
&= \sum_{y \in Y} b(y)d_X(y) + \sum_{z \in Z} b(z)d_X(z)
\end{aligned}$$

This result implies that $\sum_{y \in Y} b(y)d_X(y) \leq b(X)\Delta - b(Z)\delta$ and so provides an upperbound on the sum.

So, in a locally-fair flow, the client flow is bounded by the following inequality:

$$f_U \geq b(X) + \frac{b(Y)^2}{b(Y) + b(X)\Delta - b(Z)\delta} \quad (2)$$

Combining this result with the previous bound for the server flow, we have these two inequalities:

$$\begin{aligned}
\phi = f_U &\geq b(X) + \frac{b(Y)^2}{b(Y) + b(X)\Delta - b(Z)\delta} \\
\phi = f_V &\geq b(Y) + b(Z)\frac{\delta}{\Delta}
\end{aligned}$$

Setting $w = b(W)$, $x = b(X)$, $y = b(Y)$, and $z = b(Z)$, by Lemma 2 the value of the locally-fair maximal flow f is at least $\min(1, \frac{\Delta^2 - \delta}{2\Delta^2 - \delta\Delta - \Delta})$ times the maximum flow m and the first statement of the theorem is proved.

For the second statement of the theorem, that the given bound is tight, consider the following. For any combination of integers Δ, δ create a unit capacity network and divide the clients and servers each into two sets W, X and Y, Z . For convenience, denote $w = |W|, x = |X|, y = |Y|$, and $z = |Z|$.

Set $w = y$ and $x = z$. Let each client $u \in W$ have one outgoing edge, each client $u \in X$ have Δ outgoing edges, each server $v \in Y$ have Δ in-edges, and each server $v \in Z$ have δ in-edges. Match each client $u \in W$ to a unique server $v \in Y$ and let each client $u \in X$ have $\Delta - \delta$ edges matched to the servers in Y and the remaining δ edges matched to servers in Z such that there is a perfect matching in the graph and the non-matching edges are evenly distributed across the servers in the applicable server sets.

Given any value for y , we set $x = y \frac{\Delta-1}{\Delta-\delta}$, and the other two parameters (w and z) are determined by the equality constraints above. (The values x and y can be scaled so that they are integers.) The size of y is determined by the fact that $x + y = m$ which implies that $y = m \frac{\Delta-\delta}{2\Delta-\delta-\Delta}$. See Figure 1 which shows an example.

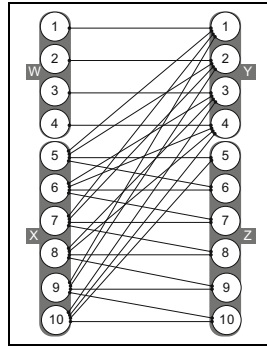


Fig. 1. Worst-case graph with $y = 4$, $\Delta = 4$, and $\delta = 2$. The maximum flow is 10 while a locally-fair maximal flow only provides $\frac{\Delta^2-\delta}{2\Delta^2-\delta-\Delta}10 = 7$.

Define a flow f such that each client $u \in W$ sends $1/\Delta$ flow along its single edge, and each client $u \in X$ sends $1/\Delta$ flow along each of its Δ edges. Clearly, f is a locally-fair flow and every server $v \in Y$ is saturated while every server $v \in Z$ is unsaturated and receives δ/Δ flow.

The maximum flow in the graph is equal to $x + y = m$, since G contains a perfect matching. Thus, the ratio of the locally-fair flow f to the maximum flow is $(y + z(\delta/\Delta))/(y + x) = (\Delta^2 - \delta)/(2\Delta^2 - \Delta - \delta\Delta)$ since $z = x$. This is equal to our lower bound and thus the second statement of the theorem is proved.

Lemma 2. *Given constants m , δ , and Δ , the solution to the non-linear program below is a value at least $\min(1, \frac{\Delta^2-\delta}{2\Delta^2-\delta-\Delta})m$.*

$$\begin{aligned} & \min f(x, y, z) \\ & \text{s.t.} \\ & f(x, y, z) \geq x + \frac{y^2}{y+x\Delta-z\delta} \\ & f(x, y, z) \geq y + z\frac{\delta}{\Delta} \end{aligned}$$

$$\begin{aligned}
m &\geq x, y \geq 0 \\
x + y &\geq m \\
y + z &\geq m \\
\Delta &> \delta \geq 1
\end{aligned}$$

Proof. The proof is given in the full version of the paper.

4 Practicality

In a network of clients and servers it is well-known that TCP/IP, or any additive-increase multiplicative-decrease (AIMD) algorithm, converges to a flow whereby each unsaturated client receives bandwidth from each server at least equal to its fair share[7]. The definition of ‘Locally-fair’ in this paper is a slightly stronger requirement since it has the additional constraint that every edge contain at least a minimal amount of flow. However, in our preliminary experiments, TCP/IP typically has higher aggregate bandwidth than the equivalent ‘locally-fair’ assignment would predict on the same network. So, it may be that the bounds in this paper are lower-bounds for max-min fair algorithms such as TCP/IP. This remains future research.

The degree-bounds in this paper are either locally enforceable or may occur naturally in practice. Networks with low Δ and high δ values can be achieved in several ways. It is common for clients to have restricted access to servers (or peers) in order to limit the advantage of any single client over any other [8]. This is a side-effect of the max-min fairness provided by TCP/IP to clients – the more streams the more bandwidth a client can steal from others. Furthermore, maintaining a set of streams becomes impractical as the number of streams gets too large: TCP/IP will reset the sender’s TCP window on any stream which is idle for more than a timeout period [9]. This effectively limits the number of outgoing streams a single modern computer can maintain. (There is no equivalent idle-detection mechanism on the receiver side.) So it is clear that limiting the maximum client degree, Δ , should be practical and simple to implement in software distributed to clients.

Maintaining the minimum server degree, δ , can be managed in one of two ways. First, in a randomly-constructed network where every client has exactly Δ connections, the minimum server degree is within a constant factor of Δ with high probability. With a balls-and-bins analysis (omitted here) it can be shown that as long as $\Delta \geq 4c^2 \ln(2|V|)$ then $\delta \geq (1 - 1/c)\Delta$ w.h.p. Thus, in a randomly-constructed network, the two parameters Δ and δ can be coupled as long as $\Delta \in \Omega(\ln(|V|))$. Another way to increase the value of δ is via a systems-level solution of server gossiping – clients can be exchanged from low-degree servers to high-degree servers by periodic server-to-server contact.

5 Open Issues and Further Research

Other types of maximal flows are also of interest: “Locally-Greedy Maximal Flows” (LGMF) and “Locally-Proportional Maximal Flows” (LPMF). In LGMF

saturating clients satisfy server demands from largest to smallest value while in LPMF these saturating clients satisfy server demands in relative proportion.

For the unit-capacity case where the maximum client degree is 2, it can be shown that LGMF always reaches at least $5/6$ of the maximum flow and LPMF always reaches at least $3/4$ of the maximum flow. (LFMF has a tight bound of $3/4$ of the maximum flow for this case.) Furthermore, these bounds are tight (proofs are omitted). For general graphs, it is an open problem to determine how well these flows perform.

References

1. Annexstein, F., Berman, K.A., Strunjas, S., Yoshikawa, C.: Maximizing Throughput in Minimum Rounds in an Application-Level Relay Service. In: Workshop on Algorithm Engineering & Experiments, ALENEX (2007)
2. Awerbuch, B., Hajiaghayi, M.T., Kleinberg, R.D., Leighton, T.: Online Client-Server Load Balancing without Global Information. In: SODA 2005: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 197–206. Society for Industrial and Applied Mathematics, Philadelphia (2005)
3. Cardinal, J., Labbé, M., Langerman, S., Levy, E., Mélot, H.: A tight analysis of the maximal matching heuristic. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 701–709. Springer, Heidelberg (2005)
4. Cherkassky, B.V., Goldberg, A.V., Martin, P., Setubal, J.C., Stolfi, J.: Augment or Push: A Computational Study of Bipartite Matching and Unit-Capacity Flow Algorithms. *J. Exp. Algorithmics* 3, 8 (1998)
5. Czygrinow, A., Hańćkowiak, M., Szymańska, E.: Distributed algorithm for approximating the maximum matching. *Discrete Appl. Math.* 143(1-3), 62–71 (2004)
6. Hanckowiak, Karonski, Panconesi: On the distributed complexity of computing maximal matchings. In: SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms) (1998)
7. Kurose, J.F., Ross, K.W.: Computer Networking: A Top-Down Approach Featuring the Internet Package. Addison-Wesley Longman Publishing Co., Inc., Boston (2000)
8. Qiu, D., Srikant, R.: Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks (2004)
9. Stevens, W.: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, Internet Engineering Task Force (January 1997)

On Finding Small 2-Generating Sets

Isabelle Fagnot¹, Guillaume Fertin², and Stéphane Vialette³

¹ IGM-LabInfo, CNRS UMR 8049, Université Paris-Est,
5 Bd Descartes 77454 Marne-la-Vallée, France
and Université Paris Diderot, Paris 7, France
`fagnot@univ-mlv.fr`

² Laboratoire d'Informatique de Nantes-Atlantique (LINA), UMR CNRS 6241
Université de Nantes, 2 rue de la Houssinière, 44322 Nantes Cedex 3 - France
`fertin@lina.univ-nantes.fr`

³ IGM-LabInfo, CNRS UMR 8049, Université Paris-Est,
5 Bd Descartes 77454 Marne-la-Vallée, France
`vialette@univ-mlv.fr`

Abstract. Given a set of positive integers S , we consider the problem of finding a minimum cardinality set of positive integers X (called a *minimum 2-generating set of S*) s.t. every element of S is an element of X or is the sum of two (non-necessarily distinct) elements of X . We give elementary properties of 2-generating sets and prove that finding a minimum cardinality 2-generating set is hard to approximate within ratio $1 + \varepsilon$ for any $\varepsilon > 0$. We then prove our main result, which consists in a representation lemma for minimum cardinality 2-generating sets.

1 Introduction

In this paper, we consider the problem of 2-generating a set of positive integers S with a minimum cardinality set of integers X , where X is said to 2-generate S if every element of S is an element of X or is the sum of two (non-necessarily distinct) elements of X . We note that, in this context, X does not have to be a subset of S . We refer to this problem as MINIMUM 2-GENERATING SET.

MINIMUM 2-GENERATING SET is a simple restriction of MINIMUM GENERATING SET (a natural problem in number theory) [4]. The MINIMUM GENERATING SET problem is defined as follows: Given a set of positive integers S , find a minimum cardinality set of integers X such that every element of S is the sum of a subset of X . MINIMUM GENERATING SET has been shown to be **NP**-hard [4], and is related, among other things, to planning radiation therapy: elements of S represent radiation dosages required at various points, while an element of X represents a dose delivered simultaneously to multiple points. Note also that both MINIMUM 2-GENERATING SET and MINIMUM GENERATING SET can be seen as natural extensions of the Postage Stamp problem [13].

Strongly related to our work are minimum sum covers of finite Abelian groups as investigated in [9,7]. A subset X of a finite Abelian group G is said to be a *sum cover* of G if $\{x + x' : x, x' \in X\} = G$, a *strict sum cover* of G if

$\{x + x' : x, x' \in X \wedge x \neq x'\} = G$, and a *difference cover* of G if $\{x - x' : x, x' \in X\} = G$. Swanson [18] gives some constructions and computational results for maximum difference packings of cyclic groups. Haanpää, Huima, and Östergård compute maximum sum and strict sum packings of cyclic groups [10]. Fitch and Jamison [7] give minimum sum and strict sum covers of small cyclic groups, and Wiedemann [19] determines minimum difference covers for cyclic groups of order at most 133.

Another area of research related to our work is the problem of covering a set of strings S with a set X of substrings in S , where X is said to *cover* S if every string in S can be written as a concatenation of the substrings in X [12,2] (see also [14] and [3] for a more general treatment of the combinatorial rank). Covering a set of strings S with a set X of substrings in S is indeed the MINIMUM GENERATING SET problem for unary alphabet under the unary encoding scheme. To narrow the context, notice that, given a set of binary strings S , finding a minimum cardinality set X of substrings in S such that every string in S can be written as a concatenation of *at most* two substrings in X is **NP**-complete (the proof being an easy binary alphabet encoding of the result of Néraud [14]). Finally, Hajiaghayi *et al.* [11] considered the MINIMUM MULTICOLORED SUBGRAPH problem, which can be seen as a generalization of MINIMUM 2-GENERATING SET when every integer in the input set is bounded by a polynomial in the length of the input.

This paper is organized as follows: we first recall basic definitions in Section 2, and we then formally introduce the considered problem. In Section 3, we give some elementary properties of 2-generating sets. Section 4 is devoted to prove hardness of MINIMUM 2-GENERATING SET and we prove in Section 5 a representation lemma. Notice that some proofs are omitted due to space constraints.

2 Preliminaries

We use $\mathbb{N}^*$ to refer to the set of all natural numbers excluding zero, *i.e.*, $\mathbb{N}^* = \{1, 2, \dots\}$. Let $S = \{s_1, s_2, \dots, s_n\} \subset \mathbb{N}^*$. For any $k \in \mathbb{N}^*$, we write kS for the set of all integers that can be expressed as the sum of *exactly* k *non necessarily distinct* integers of S , *i.e.*, $kS = \{s_{i_1} + s_{i_2} + \dots + s_{i_k} : s_{i_1}, s_{i_2}, \dots, s_{i_k} \in S\}$. According to this definition, for any set S , $S = 1S$. A set $X \subset \mathbb{N}^*$ is a *k-generating set* of S (or *k-generates* S) if $S \subseteq \bigcup_{i=1}^k iX$. (Notice here that we do not require the additional constraint $\bigcup_{i=1}^k iX \subseteq S$.) It is called a *minimum k-generating set* of S if X is a k -generating set of S of minimum cardinality. The *k-rank* of S , denoted $\text{rk}_k(S)$, is the cardinality of a minimum k -generating set of S . A set $S \subset \mathbb{N}^*$ is *k-elementary* if $\text{rk}_k(S) = |S|$. Let $\min(S)$ and $\max(S)$ stand for $\min\{s_i : s_i \in S\}$ and $\max\{s_i : s_i \in S\}$, respectively. The *length* of S , denoted $\text{len}(S)$, is defined to be $\text{len}(S) = \max(S) - \min(S)$.

We are now in position to define the MINIMUM k -GENERATING SET problem we are interested in: Given a set $S \subset \mathbb{N}^*$, find a k -generating set X of S of minimum cardinality. Actually, our main interest here is in finding minimum 2-generating sets, and hence we shall be concerned in this paper with MINIMUM 2-GENERATING SET only. Of particular importance, we assume hereafter any

reasonable (e.g. binary) encoding of any instance of MINIMUM 2-GENERATING SET so that the input is in $O(n \log m)$ space, where $n = |S|$ and $m = \max(S)$.

We assume readers have basic knowledge about graph theory [5] and we shall only recall basic notations. We write $G = (V, E)$ for a graph with *vertex set* V and *edge set* E . For a graph G and a vertex $u \in V$, we write $d_G(u)$ for the *degree* of u in G . A graph is *bipartite* if it does not contain an odd cycle.

3 Elementary Properties

Generalities. To fix the context, we begin by giving easy bounds for $\text{rk}_2(S)$.

Lemma 1. *For any $S \subset \mathbb{N}^*$ of cardinality n , $\lceil \frac{1}{2}(\sqrt{8n+9} - 3) \rceil \leq \text{rk}_2(S) \leq n$.*

Proof. The upper bound is trivial. To prove the lower bound, let $X \subset \mathbb{N}^*$ be a 2-generating set of S , and let k stand for $|X|$. For one, $|X \cup 2X| \leq \binom{k}{2} + 2k$. For another, $|X \cup 2X| \geq n$ since X 2-generates S . Combining the two inequalities yields the claimed lemma. $\square$

Combinatorial properties of intervals [8] will prove to be a simple but powerful tool for 2-generating sets. We write $[i : i+j]$ for the set of consecutive integers (*i.e.*, interval) $\{i, i+1, \dots, i+j\}$. For any interval system $\mathcal{I}$, the *matching number* of $\mathcal{I}$, denoted $\nu(\mathcal{I})$, is the maximum number of pairwise disjoint intervals of $\mathcal{I}$. Let $S = \{s_i : 1 \leq i \leq n\} \subset \mathbb{N}^*$. Define the *2-generating interval system* of S , in symbols $\mathcal{I}_2(S)$, to be $\mathcal{I}_2(S) = \{[s_i/2] : s_i \in S\}$.

Lemma 2. *Let $S \subset \mathbb{N}^*$ and $X \subset \mathbb{N}^*$ be a 2-generating set of S . Then, for every $s \in S$, $|X \cap [s/2] : s| \neq \emptyset$.*

Proof. Suppose the lemma is false. Then some $s \in S$ is obtained by summing at most 2 integers of X , each upper-bounded by $\lceil s/2 \rceil - 1$. But $2(\lceil s/2 \rceil - 1) < 2((s/2 + 1) - 1) = s$ which yields the desired contradiction. $\square$

Corollary 1. *For any $S \subset \mathbb{N}^*$, $\nu(\mathcal{I}_2(S)) \leq \text{rk}_2(S)$.*

It follows from Lemma 1 that if $\nu(\mathcal{I}_2(S)) = |S|$ then S is 2-elementary. The converse is false as shown by $S = \{7, 8, 9\}$. The following application of Corollary 1 will prove useful in the sequel.

Lemma 3. *Let $A = \{a_i : 1 \leq i \leq n\} \subset \mathbb{N}^*$ be such that (i) $a_1 \geq 4$ and (ii) $a_{i+1} > 4a_i - 3$, $1 \leq i \leq n-1$. Then, the set $S = \{2a_i - 1 : 1 \leq i \leq n\} \cup \{4a_i - 3 : 1 \leq i \leq n\} \subset \mathbb{N}^*$ is 2-elementary.*

Integer arithmetic sequences. An *integer arithmetic sequence* is a sequence of integers such that the difference of any two successive members of the sequence is a constant.

Lemma 4. *Let $S \subset \mathbb{N}^*$ be an integer arithmetic sequence of length n . Then $\text{rk}_2(S) = \Theta(\sqrt{n})$.*

Proof. Write $S = \{s_0 + ic : 0 \leq i \leq n-1\}$ for some $s_0 \in \mathbb{N}^*$ and $c \in \mathbb{N}^*$. Define $X = X_1 \cup X_2$, where $X_1 = \{s_0 + ic \lceil \sqrt{n} \rceil : 0 \leq i \leq \lceil \sqrt{n} \rceil - 1\}$, and $X_2 = \{ic : 1 \leq i \leq \lceil \sqrt{n} \rceil - 1\}$. An easy check shows that $S \subseteq X \cup 2X$, and hence X is a 2-generating set of S . Clearly, $|X_1| = \lceil \sqrt{n} \rceil$ and $|X_2| = \lceil \sqrt{n} \rceil - 1$. Therefore, $|X| \leq 2\lceil \sqrt{n} \rceil - 1 \leq 2(\sqrt{n} + 1) - 1 = 2\sqrt{n} + 1$. Combining this with Lemma 1 yields the claimed result. $\square$

In case S is an arithmetic sequence of length $n = k^2$, the above lemma reduces to $\text{rk}_2(S) \leq 2\sqrt{n} - 1$. We also observe that Lemma 4 could be an issue for dealing with dense sets. Define a set $S \subset \mathbb{N}^*$ to be ε -dense if $|S| = \varepsilon \text{len}(S)$ for some $\varepsilon > 0$. The following result is an immediate consequence of Lemma 4. We also note that the (easy) proof can be turned into an approximation algorithm with performance ratio $O(\sqrt{\varepsilon})$ for ε -dense sets.

Corollary 2. *Let $S \subset \mathbb{N}^*$ be an ε -dense set of cardinality n . Then $\text{rk}_2(S) = O(\sqrt{n/\varepsilon})$.*

Integer geometric sequences. An *integer geometric sequence* is a sequence of numbers where each term, except the first one, is found by multiplying the previous one by a fixed integer $r \geq 2$ called the *common ratio*. Results turn out to be more precise compared to arithmetic sequences.

Lemma 5. *Let $S \subset \mathbb{N}^*$ be an integer geometric sequence of length n with common ratio $r \geq 2$. Then, (i) $\text{rk}_2(S) = \lceil n/2 \rceil$ if $r = 2$ and (ii) $\text{rk}_2(S) = n$ if $r > 2$.*

Proof. A straightforward application of Corollary 1 proves (ii). To prove (i), write $S = \{s_i : 1 \leq i \leq n\}$ and $S_{\text{odd}} = \{s_i : s_i \in S \wedge i \equiv 1 \pmod{2}\}$. For one, $X = S_{\text{odd}}$ is a 2-generating set of S , and hence $\text{rk}_2(S) \leq |S_{\text{odd}}| = \lceil n/2 \rceil$. For another, $\nu(\mathcal{I}_2(S)) \geq |S_{\text{odd}}|$ since $S_{\text{odd}} \subseteq S$ and $\nu(\mathcal{I}_2(S_{\text{odd}})) = |S_{\text{odd}}|$. (The latter point follows from the fact that $s_{i+2}/2 = 2s_i > s_i$ for $1 \leq i \leq n-2$.) Combining this with Corollary 1 yields $\text{rk}_2(S) \geq |S_{\text{odd}}| = \lceil n/2 \rceil$. $\square$

Expansion and contraction. Let $S \subset \mathbb{N}^*$. For any $c \in \mathbb{N}^*$, we write $S \times c$ for the set $\{s_i c : s_i \in S\}$ and we refer to $S \times c$ as the c -*expansion* of S . Similarly, for any $c \in \mathbb{N}^*$ common divisor of S , we write S/c for the set $\{s_i/c : s_i \in S\}$ and we refer to S/c as the c -*contraction* of S .

Lemma 6 (c -expansion). *Let $S \subset \mathbb{N}^*$ and $c \in \mathbb{N}^*$. Then $\text{rk}_2(S \times c) \leq \text{rk}_2(S)$.*

Replacing S by S/c in Lemma 6 yields a formulation well-suited for contraction considerations.

Corollary 3. *Let $S \subset \mathbb{N}^*$ and $c \in \mathbb{N}^*$ be a common divisor of S . Then $\text{rk}_2(S) \leq \text{rk}_2(S/c)$.*

Lemma 7 (c -contraction). *Let $S \subset \mathbb{N}^*$ and $c \in \mathbb{N}^*$ be a common divisor of S . Then, $\text{rk}_2(S/c) = \text{rk}_2(S)$ if c is odd and $\text{rk}_2(S) \leq \text{rk}_2(S/c) \leq 2 \text{rk}_2(S)$ if c is even.*

To complement Lemma 7, we observe that we may have $\text{rk}_2(S/2c) < 2\text{rk}_2(S)$ for even c as shown in the following example.

Example 1. For any $c \in \mathbb{N}^*$, let $S = \{14c, 16c, 18c\}$. Clearly, $X = \{7c, 9c\}$ is a 2-generating set of S , and hence $\text{rk}_2(S) = 2$. But $S/2c = \{7, 8, 9\}$ has no smaller 2-generating set than itself, and hence $\text{rk}_2(S/2c) = \text{card}(S/2c) = 3$.

The upper-bound $\text{rk}_2(S/c) \leq 2\text{rk}_2(S)$ in Lemma 7 is, however, not over-estimated, as shown by the following lemma.

Lemma 8. *For any $n \in \mathbb{N}^*$, there exists a set $S \subseteq \mathbb{N}^*$ of cardinality n such that*

$$\frac{\text{rk}_2(S/2)}{\text{rk}_2(S)} = 2 - \frac{1}{n+1}.$$

Proof. Let $b > 8$ be some fixed even integer. For any $n \in \mathbb{N}^*$, let $S = \{2\} \cup \{b^i + 2 : 1 \leq i \leq n\} \cup \{2b^i + 2 : 1 \leq i \leq n\}$. We can show, using Lemma 3, that $\text{rk}_2(S) = n + 1$ and $\text{rk}_2(S/2) = 2n + 1$ (proof omitted due to space constraints.) $\square$

4 Hardness

MINIMUM GENERATING SET (*i.e.*, given a set a positive integers S , find a minimum cardinality set of integers X such that every element of S is the sum of a subset of X) was proved to be **NP**-complete in [4]. We complement this result by showing that MINIMUM 2-GENERATING SET is **APX**-hard, *i.e.*, hard to approximate within ratio $1 + \varepsilon$ for any $\varepsilon > 0$.

Proposition 1. MINIMUM 2-GENERATING SET is **APX**-hard.

Proof. We propose an L-reduction [16] from VERTEX COVER for cubic graphs: Given a cubic graph $G = (V, E)$, find a minimum cardinality vertex cover of G , *i.e.*, a subset $V' \subseteq V$ such that, for each edge $\{u, v\} \in E$, at least one of u and v belongs to V' . MINIMUM VERTEX COVER for cubic graphs is **APX**-complete [1,17]. Assume, wlog, that $V = \{1, 2, \dots, n\}$. Define the corresponding instance of MINIMUM 2-GENERATING SET by defining $S \subset \mathbb{N}^*$ to be $S = \{b^0\} \cup \{b^i : 1 \leq i \leq n\} \cup \{2b^i : 1 \leq i \leq n\} \cup \{b^0 + b^i : 1 \leq i \leq n\} \cup \{b^0 + b^i + b^j : \{i, j\} \in E\}$ for some even constant $b > 4$. We claim that there exists a vertex cover of G of cardinality at most k if and only if there exists a 2-generating set for S of cardinality at most $n + k + 1$.

($\Rightarrow$) Suppose that there exists a vertex cover $V' \subseteq V$ of cardinality k of G . Define $X \subset \mathbb{N}^*$ (actually $X \subset S$) to be $X = \{b^0\} \cup \{b^i : 1 \leq i \leq n\} \cup \{b^0 + b^i : i \in V'\}$. We claim that X is a 2-generating set for S . Since $X \subset S$ and $b^0 \in X$, it is enough to prove that, for each $\{i, j\} \in E$, $b^0 + b^i + b^j$ is 2-generated by X . Indeed, since V' is a vertex cover of G , we have $i \in V'$ or $j \in V'$ (possibly both), and if we let $\ell = i$ if $i \in V'$ and $\ell = j$ if $i \notin V'$, we have $(b^0 + b^\ell) \in X$. Therefore $b^0 + b^i + b^j$ is 2-generated by X as $(b^0 + b^\ell) + b^{\ell'}$, where $\ell' = j$ if $\ell = i$ and $\ell' = i$ otherwise.

($\Leftarrow$) Conversely, let X be a 2-generating set of S . We first note that, by integrality, $b^0 \in X$. Consider any integer $1 \leq i \leq n$, and let I_i be the interval $[b^i/2 : 2b^i]$. According to Lemma 2, $|X \cap [b^i/2 : b^i]| \geq 1$ and $|X \cap [b^i : 2b^i]| \geq 1$ since $b^i \in S$ and $2b^i \in S$. Then it follows that $|X \cap I_i| \geq 1$, and $b^i \in X$ if the inequality holds as equality. As $b > 4$, we have $2b^i < b^{i+1}/2$, $1 \leq i < n$. Then it follows that the intervals I_i , $1 \leq i \leq n$, are pairwise disjoint, and hence $|X| \geq n + 1$. Now, let $k \in \mathbb{N}^*$ be such that $|X| = n + k + 1$, and let $V' \subseteq V$ be such that $|X \cap I_i| > 1$ for every $i \in V'$. According to the above, we have $|V'| \leq k$. We now claim that V' is a vertex cover of G . Indeed, assume, aiming at a contradiction, that there exists $\{i, j\} \in E$ such that $|X \cap I_i| = 1$ and $|X \cap I_j| = 1$, and, to shorten notation, set $s = b^0 + b^i + b^j$. Then it follows that $X \cap I_i = \{b^i\}$ and $X \cap I_j = \{b^j\}$. But $s \in S$, and hence $|X \cap [s/2 : s]| \geq 1$ (Lemma 2). Furthermore, if we assume $i > j$, we have $b^i/2 < s/2$ and $s < 2b^i$, and hence $[s/2 : s] \subset I_i$, i.e., $[s/2 : s]$ is a subinterval of I_i . But $X \cap I_i = \{b^i\}$, and hence we must have $(b^0 + b^j) \in X$. This is the desired contradiction since $(b^0 + b^j) \in I_j$ and $X \cap I_j = \{b^j\}$.

We omit the easy proof that the described reduction is indeed an L-reduction. (Crucial is the fact that $|V| \leq 2|V'|$ for any vertex cover V' since G is a cubic graph.) $\square$

It remains open whether MINIMUM 2-GENERATING SET is **NP**-complete if every integer in S is bounded by a polynomial in the length of the input. Indeed, neither Proposition 1 nor the **NP**-hardness result of [4] rule out the existence of a pseudo-polynomial algorithm for MINIMUM 2-GENERATING SET. Observe that this question reduces to 2-covering a set of strings S for an unary alphabet with a set X of substrings in S , where X is said to 2-cover S if every string in S can be written as a concatenation of at most two substrings in X [12]. Approximation issues of MINIMUM 2-GENERATING SET are completely unexplored yet. Notice, however, that, as long as every integer in S is not bounded by a polynomial in the length of the input, none of the approximation results of [11,12] applies.

5 Put the Blame on $\text{rk}_2(S)$ Only

Let S be any instance of MINIMUM 2-GENERATING SET. Write $n = |S|$, $m = \max(S)$ and $k = \text{rk}_2(S)$. This section is devoted to finding a minimum cardinality 2-generating set of S (from an effective computational point of view [6,15]).

As a first attempt, let us consider the brute-force approach: generate all k -subsets X of $\{1, 2, \dots, m\}$ and check for each of them whether it 2-generates S , i.e., $S \subseteq X \cup 2X$. Correctness of this algorithm is of course immediate. There are $\binom{m}{k}$ such subsets and each subset X can be identified as a 2-generating set of S in $O(k^2 \log k)$ time (assuming a unit-cost RAM model with $\log m$ word size). Therefore, the brute-force algorithm is, as a whole, a $O(m^k k^2 \log k)$ time procedure. But m (and even $\log m$) can be arbitrarily large compared to $n = O(k^2)$ and this naturally leads us to the problem of trying to confine the seemingly inevitable combinatorial explosion of computational difficulty to a function of k only [6,15].

We prove here that such an algorithm does exist for finding a minimum cardinality 2-generating set of S . Surprisingly enough, the time complexity of the proposed algorithm turns out to be even independent of $m = \max(S)$ (again assuming a unit-cost RAM model with $\log m$ word size). The main result of this paper can be stated as follows.

Lemma 9 (representation). *Let $S = \{s_i : 1 \leq i \leq n\} \subset \mathbb{N}^*$ and write k for $\text{rk}_2(S)$. Then, there exist rationals $\alpha_{i,j} \in \{-1, -2^{-1}, 0, 2^{-1}, 1\}$, $1 \leq i \leq k$ and $1 \leq j \leq n$, such that $X = \{\sum_{j=1}^n \alpha_{i,j} s_j : 1 \leq i \leq k\}$ is a minimum cardinality 2-generating set of S .*

Before proving Lemma 9, we need a new definition that translates the problem to elementary graph theory terms. Let $S = \{s_1, s_2, \dots, s_n\}$ be a set of positive integers and $X = \{x_1, x_2, \dots, x_k\}$ be a 2-generating set for S . Define an X -realization of S to be a bipartite graph $B = (S, X, E)$ such that $d_B(s) \in \{1, 2\}$ for all $s \in S$, and

- if $d_B(s) = 1$, say $\{s, x_i\} \in E$, then $s = x_i$ or $s = 2x_i$, and
- if $d_B(s) = 2$, say $\{s, x_i\} \in E$ and $\{s, x_j\} \in E$, $x_i \neq x_j$, then $s = x_i + x_j$.

Note that, in the above definition of an X -realization, X (resp. S) is considered as a set of integers, *and* as a set of vertices in a graph. We chose not to correct this ambiguity in the rest of the paper, in order to avoid heavy notations. Besides, the context will always be clear about the fact that we are concerned with integers or vertices.

Coming back to X -realizations, it is clear that every simple cycle of B has length at least 6. (A simple cycle of length 4, say (x_1, s_1, x_2, s_2) , would result in the contradiction $s_1 = x_1 + x_2 = s_2$.) An X -realization of S is said to be *minimum* if X is a minimum cardinality 2-generating set of S . Of course, an X -realization of a set S may not be unique.

Lemma 10. *Let $S \in \mathbb{N}^*$, $B = (S, X, E)$ be a minimum X -realization of S , and let B' be any connected component of B . If $d_{B'}(s) = 2$ for every vertex $s \in S$, then there exists a simple cycle of B' of length $4\ell + 2$ for some $\ell \geq 1$.*

We are now in position to prove Lemma 9.

Proof (of Lemma 9). Write $k = \text{rk}_2(S)$. Let $X = \{x_i : 1 \leq i \leq k\}$ be a minimum cardinality 2-generating set of S and $B = (S, X, E)$ be any X -realization of S . Let $B_1, B_2, \dots, B_t$ be the connected components of B . We consider each connected component of B separately. Consider any connected component $B_i = (S_i, X_i, E_i)$ of B with $S_i \subseteq S$ and $X_i \subseteq X$. Wlog, write $S_i = \{s_1, s_2, \dots, s_{n_i}\}$. It is enough to show that for any $x \in X_i$, there exist rationals $\alpha_j \in \{-1, -2^{-1}, 0, 2^{-1}, 1\}$, $1 \leq j \leq n_i$, such that $x = \sum_{1 \leq j \leq n_i} \alpha_j s_j$, i.e., x is a linear combination with coefficients taken from $\{-1, -2^{-1}, 0, 2^{-1}, 1\}$ of the vertices in S_i . We need to consider two cases: (1) $d_{B_i}(s) = 1$ for some $s \in S_i$; or (2) $d_{B_i}(s) = 2$ for every vertex $s \in S_i$.

(1) $d_{B_i}(s) = 1$ **for some** $s \in S_i$. For convenience, write $s_1 = s$. Let P be a simple path from vertex s_1 to vertex x . (Such a path exists since B_i is connected.) Wlog, write $P = (s_1, x_1, s_2, x_2, \dots, x_{p-1}, s_p, x)$. Then it follows $s_1 = \delta x_1$, $s_2 = x_1 + x_2$, $\dots$, $s_p = x_{p-1} + x$ for some $\delta \in \{1, 2\}$, and hence $x = \frac{(-1)^{p-1}}{\delta} s_1 + \sum_{i=2}^p (-1)^{p-i} s_i$. Therefore there exist rationals $\alpha_i \in \{-1, -2^{-1}, 2^{-1}, 1\}$, $1 \leq i \leq p$, such that $x = \sum_{i=1}^p \alpha_i s_i$, i.e., x is a linear combination with coefficients taken from $\{-1, -2^{-1}, 2^{-1}, 1\}$ of those vertices s_i that lie on the path from s_1 to x .

(2) $d_{B_i}(s) = 2$ **for every vertex** $s \in S_i$. According to Lemma 10, there exists a simple cycle C of length $4\ell + 2$ for some $\ell \geq 1$ in B_i . Write $C = (x_p, s_{p+1}, x_{p+1}, \dots, x_{p+q-1}, s_{p+q})$, for some $q = 2\ell + 1$. Since graph B_i is bipartite, any cycle that starts at a vertex in S_i must alternate between vertices in S_i and X_i , and hence must be of even length (on return to the start vertex again).

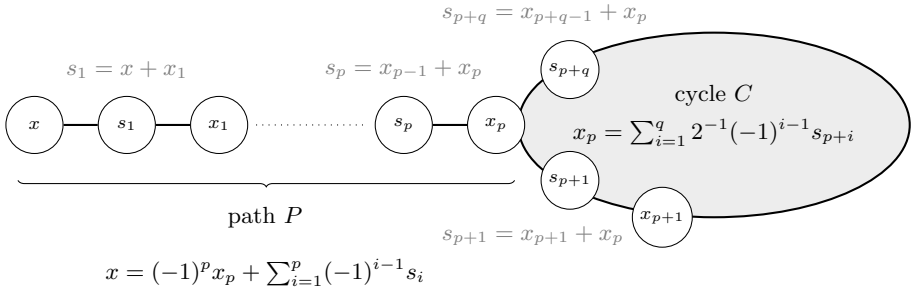


Fig. 1. For every vertex $s \in S_i$, we have $d_{B_i}(s) = 2$

For the sake of presentation, suppose first that x does not lie on cycle C (see Figure 1 for an illustration). Observe now that, since every vertex of S_i has degree 2 in B_i , every path leading from vertex x to cycle C intersects C at a vertex of X_i . Consider a shortest path leading from vertex x to cycle C , say $P = (x_0 = x, s_1, x_1, \dots, x_{p-1}, s_p, x_p)$. Note that such a path exists since B_i is connected. Clearly, since vertex x does not lie on cycle C and P is a shortest path, all vertices of P but vertex x_p do not lie on cycle C . For one, we have $s_1 = x + x_1$, $s_2 = x_1 + x_2$, $\dots$, $s_p = x_{p-1} + x_p$, and hence $x = (-1)^p x_p + \sum_{i=1}^p (-1)^{i-1} s_i$ (1). For another, $s_{p+1} = x_p + x_{p+1}$, $s_{p+2} = x_{p+1} + x_{p+2}$, $\dots$, $s_{p+q} = x_{p+q-1} + x_p$, and hence $x_p = \sum_{i=1}^q 2^{-1}(-1)^{i-1} s_{p+i}$ (2) since p is odd. Combining (1) and (2) yields $x = \sum_{i=1}^q 2^{-1}(-1)^{p+i-1} s_{p+i} + \sum_{i=1}^p (-1)^{i-1} s_i$. Therefore, there exist rationals $\alpha_i \in \{-1, -2^{-1}, 0, 2^{-1}, 1\}$, $1 \leq i \leq n$, such that $x = \sum_{i=1}^n \alpha_i s_i$. More precisely, x is a linear combination with coefficients taken from $\{-1, -2^{-1}, 2^{-1}, 1\}$ of those vertices s_i that lie on a shortest path leading from vertex x to a cycle C or lie on cycle C .

If x lies on cycle C , (1) vanishes (zero-length path), and substituting x_p by x in (2) yields $x = \sum_{i=1}^q 2^{-1}(-1)^{i-1} s_{p+i}$. Therefore, x is a linear combination with coefficients taken from $\{-2^{-1}, 2^{-1}\}$ of those vertices s_i that lie on cycle C . $\square$

Thanks to Lemma 9, we can prove that there exists an algorithm for MINIMUM 2-GENERATING SET that confines the combinatorial explosion of computational difficulty to a function of $k = \text{rk}_2(S)$ only.

Proposition 2. *Assuming a unit-cost RAM model with $\log m$ word size ($m = \max(S)$), there exists a $O(5^{\frac{k^2(k+3)}{2}} k^2 \log k)$ time algorithm for finding a minimum cardinality 2-generating set of S , where $k = \text{rk}_2(S)$.*

Proof. We propose a brute-force algorithm for finding a (representation of a) minimum cardinality 2-generating set of S . The basic idea is to consider the set $C(S)$ of all linear combinations $\alpha_1 s_1 + \alpha_2 s_2 + \dots + \alpha_n s_n$ with coefficients taken from $\{-1, -2^{-1}, 0, 2^{-1}, 1\}$. Clearly, there exist 5^n such combinations. The algorithm simply tries each k -subset X of $C(S)$ and checks whether $S \subseteq X \cup 2X$. Correctness of the algorithm follows from Lemma 9. We now turn to proving its time complexity. Let N be the number of k -subsets of $C(S)$. Clearly, $N = \binom{5^n}{k} = O(5^{nk})$. But $n \leq \frac{k(k+3)}{2}$, and hence $N = O(5^{\frac{k^2(k+3)}{2}})$. Since a k -subset X of $C(S)$ can be identified as a 2-generating set of S in $O(k^2 \log k)$ time (assuming a unit-cost RAM model with $\log m$ word size), the total running time is $O(Nk^2 \log k) = O(5^{\frac{k^2(k+3)}{2}} k^2 \log k)$. $\square$

6 Conclusion

MINIMUM 2-GENERATING SET is a natural restriction of MINIMUM GENERATING SET with prospective applications (see [4]). Our representation (Lemma 9) provides a first positive algorithmic result for computing minimum 2-generating sets. We mention here some directions of interest for future works: (1) Is MINIMUM 2-GENERATING SET pseudo-polynomial time solvable? Notice that this question is related to 2-covering a set of strings S for a unary alphabet with a set X of substrings in S , where X is said to 2-cover S if every string in S can be written as a concatenation of at most two substrings in X [12]. (2) For any $k > 1$, a set of integers S is said to be k -simplifiable if $\text{rk}_k(S) < |S|$ [14]. Is there a polynomial-time algorithm for deciding whether S is 2-simplifiable? (3) Considering the general MINIMUM k -GENERATING SET problem, is there an analog of Lemma 9 for every fixed $k \geq 2$?

Acknowledgments

The authors are thankful to Olivier Serre for helpful discussions. They are also indebted to the reviewers for a careful and thoughtful reading of the original version of this paper.

References

1. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. *Theoretical Computer Science* 237(1-2), 123–134 (2000)
2. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hallett, M.T., Wareham, H.T.: Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences* 11, 49–57 (1995)
3. Choffrut, C., Karhumäki, J.: Combinatorics of words. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of formal languages, Word, language, grammar*, vol. 1, pp. 329–438. Springer, Heidelberg (1997)
4. Collins, M.J., Kempe, D., Saia, J., Young, M.: Nonnegative integral subset representations of integer sets. *Information Processing Letters* 101(3), 129–133 (2007)
5. Diestel, R.: *Graph theory*, 2nd edn. Graduate texts in Mathematics, vol. 173. Springer, Heidelberg (2000)
6. Downey, R., Fellows, M.: *Parameterized complexity*. Springer, Heidelberg (1999)
7. Fitch, M.A., Jamison, R.E.: Minimum sum covers of small cyclic groups. *Congressus Numerantium* 147, 65–81 (2000)
8. Gyárfás, A.: Combinatorics of intervals, preliminary version. In: Institute for Mathematics and its Applications (IMA) Summer Workshop on Combinatorics and Its Applications (2003), <http://www.math.gatech.edu/news/events/ima/newag.pdf>
9. Haanpää, H.: Minimum sum and difference covers of abelian groups. *Journal of Integer Sequences* 7(2), article 04.2.6 (2004)
10. Haanpää, H., Huima, A., Östergård, P.R.J.: Sets in $\mathbb{Z}_n$ with distinct sums of pairs. *Discrete Applied Mathematics* 138(1-2), 99–106 (2004)
11. Hajiaghayi, M., Jain, K., Lau, L., Russell, A., Mandoiu, I., Vazirani, V.: Minimum multicolored subgraph problem in multiplex PCR primer set selection and population haplotyping. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) *ICCS 2006*. LNCS, vol. 3994, pp. 758–766. Springer, Heidelberg (2006)
12. Hermelin, D., Rawitz, D., Rizzi, R., Vialette, S.: The minimum substring cover problem. In: Kaklamanis, C., Skutella, M. (eds.) *WAOA 2007*. LNCS, vol. 4927, pp. 170–183. Springer, Heidelberg (2008)
13. Moser, L.: On the representation of $1, 2, \dots, n$ by sums. *Acta Arithmetica* 6, 11–13 (1960)
14. Néraud, J.: Elementariness of a finite set of words is coNP-complete. *Theoretical Informatics and Applications* 24(5), 459–470 (1990)
15. Niedermeier, R.: *Invitation to fixed parameter algorithms*. Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford (2006)
16. Papadimitriou, C.H.: *Computational complexity*. Addison-Wesley, Reading (1994)
17. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation and complexity classes. *Journal of Computer and System Sciences* 43, 425–440 (1991)
18. Swanson, C.N.: Planar cyclic difference packings. *Journal of Combinatorial Designs* 8, 426–434 (2000)
19. Wiedemann, D.: Cyclic difference covers through 133. *Congressus Numerantium* 90, 181–185 (1992)

Convex Recoloring Revisited: Complexity and Exact Algorithms

Iyad A. Kanj^{1,*} and Dieter Kratsch²

¹ School of Computing, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604, USA

`ikanj@cs.depaul.edu`

² LITA, Université Paul Verlaine Metz, 57045 Metz Cedex 01, France
`kratsch@univ-metz.fr`

Abstract. We take a new look at the CONVEX PATH RECOLORING (CPR), CONVEX TREE RECOLORING (CTR), and CONVEX LEAF RECOLORING (CLR) problems through the eyes of the INDEPENDENT SET problem. This connection allows us to give a complete characterization of the complexity of all these problems in terms of the number of occurrences of each color in the input instance, and consequently, to present simpler NP-hardness proofs for them than those given earlier. For example, we show that the CLR problem on instances in which the number of leaves of each color is at most 3, is solvable in polynomial time, by reducing it to the INDEPENDENT SET problem on chordal graphs, and becomes NP-complete on instances in which the number of leaves of each color is at most 4.

This connection also allows us to develop improved exact algorithms for the problems under consideration. For instance, we show that the CPR problem on instances in which the number of vertices of each color is at most 2, denoted 2-CPR, proved to be NP-complete in the current paper, is solvable in time $2^{n/4} n^{O(1)}$ (n is the number of vertices on the path) by reducing it after $2^{n/4}$ enumerations to the weighted independent set problem on interval graphs, which is solvable in polynomial time. Then, using an exponential-time reduction from CPR to 2-CPR, we show that CPR is solvable in time $2^{4n/9} n^{O(1)}$. We also present exact algorithms for CTR and CLR running in time $2^{0.454n} n^{O(1)}$ and $2^{n/3} n^{O(1)}$, respectively.

1 Introduction

Given a tree T and a color function $\mathcal{C}$ assigning each vertex in T a color, the coloring $\mathcal{C}$ is said to be *convex* if, for each color $c \in \mathcal{C}$, the vertices of color c induce a subtree of T . The CONVEX TREE RECOLORING (CTR) problem is: given a tree T and a coloring $\mathcal{C}$ —which is not necessarily convex, recolor the minimum number of vertices in T so that the resulting coloring is convex. The CTR problem has received considerable attention in the last few years [2–5, 11–13].

The CTR problem was first studied by Moran and Snir in [11] (journal version), and was motivated by applications in computational biology (see [11] for

* Corresponding author.

an extensive discussion on these applications). Moran and Snir [11] studied the problem from within a general model in which recoloring a vertex is associated with a nonnegative cost, and a convex recoloring that minimizes the total cost is sought. This model is referred to as the *weighted model*, whereas the model in which no weights are assigned (or all weights are the same) is referred to as the *unweighted model*. For the unweighted model, it was shown in [11] that the problem is NP-hard even for the simpler case when the tree is a path, referred to as the CONVEX STRING RECOLORING problem in [11], and as the CONVEX PATH RECOLORING (CPR) problem in this paper. Moran and Snir [11] also considered the CONVEX LEAF RECOLORING (CLR) problem, where only the leaves of T are assigned colors by $\mathcal{C}$, and a convex recoloring of T that minimizes the number of recolored leaves is sought. It was also shown in [11] that CLR is NP-hard under the unweighted model. The paper [11] also studied algorithms for solving CPR and CTR under the weighted model. The authors of [11] gave an algorithm that solves CPR in time $O(n \cdot n_c \cdot 2^{n_c})$, where n_c is the number of colors and n is the number of vertices on the path. They showed that this algorithm can be extended to trees to solve CTR in time $O(n \cdot n_c \cdot \Delta^{n_c})$, where Δ is the maximum degree of the tree. Moreover, they showed that the number of colors n_c in the previously mentioned algorithms, can be replaced with the number β of bad colors: those are the colors in $\mathcal{C}$ that do not induce subtrees (or subpaths in the case of a path) in T .

From the parameterized complexity perspective, the CTR problem received a lot of attention [2–5, 11–13] under both the weighted and the unweighted models. It was studied with respect to different parameters, including the number of colors n_c , the number of bad colors β , and the number of recolored vertices k , and was shown to be fixed-parameter tractable with respect to all these parameters [2, 4, 11, 12]. Among the notable FPT results for CTR we mention the $O(2^k kn + n^2)$ time algorithm in [2], the $2^{\beta n^{O(1)}}$ time algorithm in [12], and the $O(k^2)$ kernel upper bound (on the number of vertices) in [3].

In this paper we consider the CPR, CTR, and CLR problems under the unweighted model. We take a new look at these problems through the eyes of the INDEPENDENT SET problem. This connection to INDEPENDENT SET, together with the structural results developed in this paper, allow us to give a complete characterization of the complexity of each of these three problems with respect to the maximum number of occurrences of each color in the input instance. For example, via a simple (polynomial-time) reduction from INDEPENDENT SET, we show that the CPR problem on instances in which the number of vertices of each color is at most 2, denoted 2-CPR, remains NP-complete. (Note that the CPR problem is obviously solvable in polynomial time on instances in which the number of occurrences of each color is 1.) This provides a simpler NP-hardness proof for CPR than that in [11], and characterizes the complexity of the problem modulo the number of occurrences of each color. We note that the reduction in [11] from 3-SAT to CPR produces instances in which the number of vertices of a given color is unbounded. Note also that the complexity results for CPR carry over to CTR (every path is a tree). For CLR, we show that it is solvable

in polynomial time when the number of leaves of each color is at most 3, denoted 3-CLR, by reducing it to the weighted INDEPENDENT SET problem on chordal graphs, which is solvable in polynomial time [7]. On the other hand, we show that CLR becomes NP-complete when the number of vertices of each color is at most 4, by a reduction from INDEPENDENT SET.

In addition, this connection to INDEPENDENT SET allows us to develop (moderately exponential-time) exact algorithms for CPR, CTR, and CLR, by reducing them to the INDEPENDENT SET problem on restricted classes of graphs. For example, we show that the 2-CPR problem is solvable in time $2^{n/4}n^{O(1)}$, where n is the number of vertices in the path, by reducing the problem after $O(2^{n/4})$ enumerations to the weighted INDEPENDENT SET problem on interval graphs, which is polynomial-time solvable [8]. This algorithm for 2-CPR, coupled with an exponential-time reduction from CPR to 2-CPR, give an exact algorithm for CPR running in time $2^{4n/9}n^{O(1)}$. Similarly for CTR, we can show that it can be solved in time $2^{0.454n}n^{O(1)}$. For CLR, the polynomial-time solvability for 3-CLR proved in this paper, together with an exponential time reduction from CLR to 3-CLR, give an algorithm running in time $2^{n/3}n^{O(1)}$.

We note that all the above algorithms have a better worst-case running time than the best (moderately exponential-time) exact algorithm that can be obtained as a direct byproduct of the existing parameterized algorithms. In particular, the running time of the presented algorithms is better than the $2^{n/2}n^{O(1)}$ time algorithm obtained from the $2^\beta n^{O(1)}$ time algorithm of Ponta et al. [12], after observing that the number of bad colors β is bounded by $n/2$.

2 Preliminaries

We assume familiarity with the basic graph terminologies and notations, and we refer the reader to [9] for some of the facts and results on special graph classes that are used in this paper.

For an asymptotically positive integer-function $t(n)$, we will use the asymptotic notation $O^*(t(n))$ to denote time complexity of the form $O(t(n).p(n))$, where $p(n)$ is a polynomial.

Let T be a tree, and let $\mathcal{C}$ be a function assigning each vertex in T a color in $\{c_1, \dots, c_p\}$. The coloring $\mathcal{C}$ is said to be *convex*, if for every color $c \in \{c_1, \dots, c_p\}$, the set of vertices in T of color c induces a subtree of T . If T is given with a coloring $\mathcal{C}$ of its vertices, we can view any other coloring $\mathcal{C}'$ of $V(T)$ as a recoloring of $\mathcal{C}$. For a vertex $v \in V(T)$, we say that $\mathcal{C}'$ *retains* the color of v if $\mathcal{C}'(v) = \mathcal{C}(v)$; otherwise, we say that $\mathcal{C}'$ *recolors* v . If c is a color assigned by $\mathcal{C}$, we say that the recoloring $\mathcal{C}'$ *retains* the color c if there exists at least one vertex $v \in T$ such that $\mathcal{C}'(v) = c$.

The CONVEX TREE RECOLORING problem, abbreviated CTR, is defined as follows: Given a tree T on n vertices and a function $\mathcal{C}$ assigning each vertex in T a color in $\{c_1, \dots, c_p\}$, compute a convex recoloring of T that recolors the minimum number of vertices.

The CONVEX PATH RECOLORING problem, abbreviated CPR, is defined as follows: Given a path P on n vertices and a function $\mathcal{C}$ assigning each vertex

in P a color in $\{c_1, \dots, c_p\}$, compute a convex recoloring of P that recolors the minimum number of vertices.

The CONVEX LEAF RECOLORING problem, abbreviated CLR, is defined as follows: Given a tree T with n leaves and a function $\mathcal{C}$ assigning each leaf in T a color in $\{c_1, \dots, c_p\}$, compute a convex recoloring of T that recolors the minimum number of leaves.

3 The CPR Problem

In this section we show that the CPR problem, even when restricted to instances having at most two vertices from each color, denoted by the 2-CPR problem, remains NP-complete. This provides an alternative proof of the NP-completeness of CPR. We also give an exact algorithm for the 2-CPR problem. This algorithm for 2-CPR, together with an exponential-time reduction from CPR to 2-CPR, yield an exact algorithm for CPR with the best running time.

By a reduction from INDEPENDENT SET, we can show the following theorem:

Theorem 3.1. *The 2-CPR problem is NP-complete.*

Corollary 3.1. *The CPR problem is NP-complete.*

The results in this section rely heavily on the following key structural result whose proof is omitted for lack of space:

Lemma 3.1 (The Exchange Lemma). *Let $(P, \mathcal{C})$ be an instance of 2-CPR, and suppose that there exists a convex recoloring of P that recolors at most ℓ vertices. Then there exists a convex recoloring of P that recolors at most ℓ vertices and that retains every color assigned by $\mathcal{C}$.*

Let $(P, \mathcal{C})$ be an instance of 2-CPR. Call a color c a *singleton color* if $\mathcal{C}(v) = c$ for exactly one vertex $v \in P$; otherwise, call c an *interval color*. If c is an interval color, we call the two vertices on P with color c *mates*. For two mate vertices u and v on P , we call the subpath between u and v on P an *interval*, and refer to it by $[u, v]$. We call an interval $[x, y]$ a *long interval* if there exists an interval $[u, v]$ such that $[u, v]$ is contained in $[x, y]$ (i.e., the path from u to v is a subpath of that from x to y), or if there exists a vertex w contained in $[x, y]$ such that $\mathcal{C}(w)$ is a singleton color; otherwise, we call $[x, y]$ a *short interval*. Let N_{short} be the number of short intervals on P , N_{long} that of long intervals, and note that $N_{\text{short}} + N_{\text{long}} \leq n/2$, since the total number of intervals on P is at most $n/2$.

By **The Exchange Lemma** (Lemma 3.1), we may assume that an optimal convex recoloring of P (i.e., a convex recoloring of P that recolors the minimum number of vertices), $\mathcal{C}_{\text{opt}}$, retains every color assigned by $\mathcal{C}$. In particular, for every vertex v on P whose color is a singleton color, we have $\mathcal{C}_{\text{opt}}(v) = \mathcal{C}(v)$. Moreover, for every long interval $[x, y]$, exactly one of its endpoints will retain its color by $\mathcal{C}_{\text{opt}}$. This is true because at least one of the endpoints of the interval $[x, y]$ retains its color by $\mathcal{C}_{\text{opt}}$, and at most one endpoint of $[x, y]$ retains its color (otherwise, there exists an interval $[u, v]$ inside $[x, y]$ such that none of

the vertices in $\{u, v\}$ retains its color by $\mathcal{C}_{opt}$). Therefore, when enumerating recolorings of P , we are only interested in those that retain every color on P , and we may work under this assumption. Based on the above discussion, and using the **The Exchange Lemma**, we can prove the following lemma:

Lemma 3.2. *An optimal convex recoloring for $(P, \mathcal{C})$ can be computed in time $O^*(2^{N_{short}})$.*

Lemma 3.3. *An optimal convex recoloring for $(P, \mathcal{C})$ can be computed in time $O^*(2^{N_{long}})$.*

Proof. For each long interval $[x, y]$, we enumerate which vertex in $\{x, y\}$ retains its color in an optimal convex recoloring of P . Note that exactly one vertex in $\{x, y\}$ retains its color, under the assumption that every color on P is retained by the desired enumerated recoloring. If x (resp. y) retains its color, we keep x (resp. y) and remove y (resp. x), because y (resp. x) needs to be recolored, and we can always recolor it with the color of a neighboring vertex, once a convex recoloring of the resulting path has been computed. Note that this procedure might result in a vertex whose color becomes a singleton color (because its color is retained, while its mate in a long interval needs to be recolored and has been removed). For each vertex whose color becomes a singleton color after the above enumeration, and for each interval containing it, we change the status of this interval and make it a long interval. We note that no long interval at this point contains another interval; this property will be crucial for the remaining part of the proof to go through.

Let P' be the resulting path at the end of this process. Then P' consists of: (1) vertices whose colors are singleton colors, (2) long intervals which contain at least one vertex of a singleton color but no nested intervals, and (3) short intervals. Let N'_{short} be the number of short intervals on P' , and N'_{long} that of long intervals. Note that if a convex recoloring of P' (that retains every color) retains the color of both endpoints of exactly k short intervals, then the total number of vertices on P' that need to be recolored by this recoloring is exactly $N'_{long} + N'_{short} - k$. Note also that no two short intervals whose colors are retained by a convex recoloring can contain both endpoints of another interval (otherwise, its color would not be retained), and no two such short intervals can intersect. This implies that the set of short intervals whose colors are retained by a convex recoloring of P' (that retains every color) corresponds to an independent set of the same size in the square of the interval graph $G_{P'}$, defined naturally as follows: For every short interval on P' associate a vertex in $G_{P'}$ of weight 1. For every long interval on P' associate a vertex of weight 0. Two vertices in $G_{P'}$ are adjacent if and only if their corresponding intervals on P' intersect.

If a convex recoloring $\mathcal{C}'$ of P' retains the colors of both endpoints of k short intervals on P' , and hence recolors exactly $N'_{long} + N'_{short} - k$ vertices on P' , then it is easy to verify that these k short intervals correspond to an independent set of weight k in the square of $G_{P'}$. On the other hand, if I is an independent set in $G'_{P'}$ of weight k , then I contains exactly k vertices, each of weight 1, whose corresponding short intervals on P' are of pairwise distance at least 2 (i.e., no

two short intervals on P' whose corresponding vertices are in I' intersect, and no interval on P' intersects with two intervals whose corresponding vertices are in I). Therefore, by retaining the color of every short interval corresponding to a vertex in I , and recoloring exactly one endpoint from every other interval on P' , we obtain a convex recoloring of P' that recolors $N'_{long} + N'_{short} - k$ vertices on P' . This shows that an optimal convex recoloring of P' corresponds to a maximum-weight independent set in the square of $G_{P'}$, and vice versa. Since the square of an interval graph is also an interval graph [1, 10], and since the WEIGHTED INDEPENDENT SET problem is solvable in polynomial time on interval graphs [8], computing an optimal convex recoloring after guessing the status of the long intervals on P can be done in polynomial time.

The analysis of the number of enumerations is analogous to that of Lemma 3.2. We conclude that an optimal convex recoloring for $(P, \mathcal{C})$ can be computed in time $O^*(2^{N_{long}})$. This completes the proof. $\square$

Theorem 3.2. *The 2-CPR problem can be solved in time $O^*(2^{n/4})$.*

Proof. This follows from Lemma 3.2 and Lemma 3.3, and the fact that $N_{short} + N_{long} \leq n/2$ (hence, either N_{short} or N_{long} is at most $n/4$). $\square$

We now use Theorem 3.2 to design an exact algorithm for CPR.

Let $(P, \mathcal{C})$ be an instance of CPR. Recall that a color c is *bad* [11, 12] if the vertices on P of color c do not form a subpath of P . Clearly, no singleton color is a bad color, and hence, the number of bad colors N_{bad} is at most $n/2$.

The notion of a bad color was defined for the CTR problem (by replacing subpath with subtree). The results in [12] showed that the CTR problem can be solved in time $O^*(2^{N_{bad}}) = O^*(2^{n/2})$. This shows that the CPR problem can be solved in $O^*(2^{n/2})$ time as well. We shall improve on this upper bound by reducing the CPR problem to the 2-CPR problem, and then using Theorem 3.2.

Let N_1 be the number of singleton colors on P , N_2 that of interval colors, and $N_{>2}$ that of colors that appear at least three times on P (i.e., appear on at least 3 vertices on P). For each color c that appears at least three times on P , we fix any two vertices of color c on P and call them *stationary* vertices, and we call each other vertex of color c an *excess* vertex. Let N_e be the number of excess vertices on P , and note that $N_{>2} \leq N_e$. We have the following equality:

$$N_1 + 2N_2 + 2N_{>2} + N_e = n. \quad (1)$$

Consider the following algorithm $\mathcal{A}$ that solves the CPR problem. For every excess vertex, enumerate whether the color of this vertex is retained by an optimal convex recoloring of P or not. In addition, for each color that appears at least 3 times on P , pick one stationary vertex of that color (arbitrarily) and enumerate whether the color of that vertex is retained or not by an optimal convex recoloring of P . Note that, if, for a color c , two vertices of color c that are retained by this enumeration are separated by a vertex of color c whose status is not retained, then we can reject this enumeration. Moreover, no two vertices of the same color, whose color is retained, can be separated by a vertex of different

color whose color is either retained or is a singleton color; otherwise, we reject the enumeration.

For each vertex whose color is not retained, we remove the vertex because its color can be determined once an optimal convex recoloring has been computed (such a vertex can be recolored with the color of one of its neighbors). Afterwards, for each color c such that at least two vertices of color c retain their color under the enumeration, we pick the two vertices of color c that are farthest apart on P , color all the vertices between them with color c , and shrink all these vertices to a single vertex whose color becomes a singleton color. After this operation, no color appears more than twice on P , and we end up with an instance $(P', \mathcal{C}')$ of the 2-CPR problem; let n' be the number of vertices on P' .

The number of vertices enumerated by the above procedure is $N_e + N_{>2}$. Therefore, the total number of enumerations in the above procedure is bounded by $2^{N_e + N_{>2}} \leq 2^{2N_e}$. After that, we apply Theorem 3.2 to the instance $(P', \mathcal{C}')$, whose number of vertices n' is at most $n - N_e$ (at most 2 vertices from each color remain), which takes time $O^*(2^{(n-N_e)/4})$. Therefore, algorithm $\mathcal{A}$ solves the CPR problem in time $O^*(2^{(n+7N_e)/4})$.

On the other hand, the number of bad colors is $N_2 + N_{>2} \leq (n - N_e)/2$ by Equation (1). Therefore, using the results in [12], we can solve the CPR problem by an algorithm $\mathcal{A}'$ in time $O^*(2^{(n-N_e)/2})$. This suggests the following algorithm. If $N_e > n/9$, we apply the algorithm $\mathcal{A}'$ that solves the CPR problem in time $O^*(2^{4n/9})$, and if $N_e \leq n/9$, we apply the algorithm $\mathcal{A}$ which solves the CPR problem in time $O^*(2^{4n/9})$. Therefore, we have the following theorem:

Theorem 3.3. *The CPR problem is solvable in time $O^*(2^{4n/9})$.*

4 The CTR Problem

We define the 2-CTR problem to be the set of instances $(T, \mathcal{C})$ of CTR such that, for every color c , the number of vertices in T of color c is at most 2. The NP-completeness of 2-CTR, and hence of CTR, follows from the NP-completeness of 2-CPR established by Theorem 3.1, since a path is a special case of a tree.

Let $(T, \mathcal{C})$ be an instance of 2-CTR. The general approach we use for designing an exact algorithm for 2-CTR is very similar to that used for 2-CPR. First, **The Exchange Lemma** (Lemma 3.1) for 2-CPR carries over to 2-CTR. Second, we can define the notions of short and long intervals similarly. Here the interval determined by two vertices on T that are mates is defined to be the unique path in T between the two vertices. The statement of Lemma 3.2 carries over: If N_{short} is the number of short intervals on T , then a similar proof to that of Lemma 3.2 shows that 2-CTR is solvable in time $O^*(2^{N_{short}})$.

Unfortunately, the statement of Lemma 3.3 does not carry over to 2-CTR. The reason being that the auxiliary graph $G_{T'}$ ($G_{P'}$ in the case of 2-CPR) is no longer an interval graph, and using a polynomial-time algorithm for computing a maximum weighted independent set on the square of $G_{T'}$ is no longer an option¹. We deal with this issue by changing the definition of $G_{T'}$, and reducing

¹ It can be proved that INDEPENDENT SET is NP-complete on squares of chordal graphs.

the problem to that of computing a maximum (in terms of cardinality) independent set on the newly-defined auxiliary graph $G_{T'}$. To compute a maximum independent set in $G_{T'}$, we now use one of the exact (exponential-time) algorithms for computing a maximum independent set on a general graph, namely the polynomial-space algorithm by Fomin et al. [6], which computes a maximum independent set in a graph of N vertices in time $O^*(2^{0.288N})$. We present the modified version of Lemma 3.3 next.

Lemma 4.1. *An optimal convex recoloring for $(T, \mathcal{C})$ can be computed in time $O^*(2^{0.712N_{long}+0.144n})$.*

Proof. For each long interval, we start by enumerating which of its endpoints retains its color. The endpoint that does not retain its color is removed. The endpoint retaining its color remains, and becomes a vertex whose color is a singleton color; note again that every short interval containing such a vertex becomes a long interval.

Let T' be the tree resulting from T at the end of this enumeration. We define the auxiliary graph $G_{T'}$ as follows. For each short interval in T' , we correspond a vertex in $G_{T'}$. Two vertices u and v in $G_{T'}$ are adjacent if and only if: (1) their corresponding short intervals in T' intersect, or (2) there exists an interval $[x, y]$ in T' such that each of the intervals corresponding to u and v in T' contains one endpoint of $[x, y]$. By a similar argument to that made in Lemma 3.3, it can be shown that a maximum independent set in $G_{T'}$ corresponds to an optimal convex recoloring of T' .

Enumerating all long intervals takes $O^*(2^{N_{long}})$ time. Computing a maximum independent set in $G_{T'}$ takes time $O^*(2^{0.288N_{short}})$ using the algorithm in [6], after noting that the number of vertices in $G_{T'}$ is at most N_{short} .

Therefore, an optimal convex recoloring for $(T, \mathcal{C})$ can be computed in time $O^*(2^{N_{long}+0.288N_{short}})$. The statement of the lemma follows after noting that $N_{long} + N_{short} \leq n/2$. $\square$

Theorem 4.1. *The 2-CTR problem can be solved in time $O^*(2^{0.293n})$.*

Proof. If $N_{short} \leq 0.293n$, then the statement follows from the fact that 2-CTR is solvable in time $O^*(2^{N_{short}})$ (see the discussion at the beginning of this section). Otherwise, from the fact that $N_{short} + N_{long} \leq n/2$, we derive $N_{long} \leq 0.207n$. Now the statement follows from Lemma 4.1. $\square$

Now to design an exact algorithm for CTR, we reduce CTR to 2-CTR, and then to use Theorem 4.1. The reduction is very similar to that from CPR to 2-CPR, described in Section 3. We can derive the following theorem:

Theorem 4.2. *The CTR problem is solvable in time $O^*(2^{0.454n})$.*

5 The CLR Problem

In this section we show that 3-CLR is solvable in polynomial time. We then use this fact to develop an exact algorithm for the CLR problem by reducing it to 3-CLR. We can also show:

Theorem 5.1. *The 4-CLR problem is NP-complete.*

Here 4-CLR consists of the set of instances of CLR in which each color appears on at most 4 leaves; the proof is omitted for lack of space.

Let $(T, \mathcal{C})$ be an instance of CLR. We first observe a stronger version of **The Exchange Lemma** for CLR, than those for 2-CPR and 2-CTR.

Observation 5.1. *Let $\mathcal{C}_{opt}$ be a convex recoloring of $\mathcal{C}$ that recolors the minimum number of leaves in T . Then $\mathcal{C}_{opt}$ retains every color in $\mathcal{C}$.*

Theorem 5.2. *The 3-CLR problem is solvable in polynomial time.*

Proof. Let $(T, \mathcal{C})$ be an instance of 3-CLR, and let n_c be the number of colors that appear in T . Call a color that appears on 3 leaves in T a *tripodal* color. We construct a graph G_T as follows.

For each interval color c that appears on two leaves x and y in T , we associate a vertex v_{xy} of weight 1 in G_T that we call a *path vertex*. For each tripodal color c that appears on three leaves x , y , and z in T , we associate 4 vertices in G_T : three path vertices v_{xy} , v_{xz} , and v_{yz} each of weight 1, and a vertex v_{xyz} of weight 2 that we call a *tripodal vertex*. Two vertices in G_T are adjacent if and only if their corresponding subtrees intersect (for a path vertex, the corresponding subtree is the path in T between the two corresponding leaves, and for a tripodal vertex, the corresponding subtree is the subtree of T that is the union of the three paths corresponding the three possible combinations of the corresponding 3 leaves). This completes the construction of G_T .

Observe that the vertices corresponding to the same tripodal color form a clique in G_T . Observe also that G_T , which is the intersection graph of some subtrees of T , is a chordal graph.

Let $\mathcal{C}_{opt}$ be an optimal convex recoloring of the leaves in T . By Observation 5.1, $\mathcal{C}_{opt}$ retains every color. Therefore, the number of leaves retaining their colors by $\mathcal{C}_{opt}$ can be expressed as $n_c + k$, for some non-negative integer k , and equivalently, the number of leaves that are recolored by $\mathcal{C}_{opt}$ can be expressed as $n - n_c - k$. From each color c appearing in T , fix one leaf of color c that retains its color by $\mathcal{C}_{opt}$. Therefore, there are precisely k leaves other than the fixed ones, whose colors are retained by $\mathcal{C}_{opt}$; call each of these leaves a *floating* leaf. We construct a set of vertices I in G_T as follows. For each color c such that there is exactly one floating leaf of color c , this floating leaf, together with the fixed leaf of color c correspond to a path vertex in G_T of weight 1: we place this vertex in I . For each floating color c such that there are exactly two floating leaves of color c , these two floating leaves, together with the fixed leaf of color c , correspond to a tripodal vertex in G_T of weight 2: we place this vertex in I . Observe that the total weight of the vertices in I is k . Moreover, the set of vertices I is an independent set in G_T , since each vertex in I corresponds to a subtree of T whose vertices all have the same unique color in $\mathcal{C}_{opt}$ (by the convexity of $\mathcal{C}_{opt}$). It follows that G_T has an independent set of weight k .

Conversely, if I is an independent set in G_T of weight k , then the recoloring that retains the colors of the leaves corresponding to the vertices in I , and retains

the color of exactly one leaf from every other color in T , recolors exactly $n - n_c - k$ leaves, and can be extended to a convex recoloring of T .

This shows that an optimal convex recoloring of $(T, \mathcal{C})$ can be computed by computing a maximum weighted independent set in the chordal graph G_T . Since computing a maximum weighted independent set in a chordal graph can be done in polynomial time [7], the theorem follows. $\square$

Using Theorem 5.2, and a similar approach to that used in Section 4, we get:

Theorem 5.3. *The CLR problem is solvable in time $O^*(2^{n/3})$.*

Acknowledgment

We would like to thank Andreas Brandstädt, Haiko Müller, and Oriana Ponta for the helpful discussions related to some of the problems in this paper.

References

1. Agnarsson, G., Greenlaw, R., Halldórsson, M.: On powers of chordal graphs and their colorings. *Congressus Numerantium* 144, 41–65 (2000)
2. Bar-Yehuda, R., Feldman, I., Rawitz, D.: Improved approximation algorithm for convex recoloring of trees. *Theory of Computing Systems* 43(1), 3–18 (2008)
3. Bodlaender, H., Fellows, M., Langston, M., Ragan, M., Rosamond, F., Weyer, M.: Quadratic kernelization for convex recoloring of trees. In: Lin, G. (ed.) *COCOON 2007*. LNCS, vol. 4598, pp. 86–96. Springer, Heidelberg (2007)
4. Bodlaender, H., Weyer, M.: Convex and connected recoloring of trees and graphs (unpublished manuscript, 2005)
5. Chor, B., Fellows, M., Ragan, M., Razgon, I., Rosamond, F., Snir, S.: Connected coloring completion for general graphs: Algorithms and complexity. In: Lin, G. (ed.) *COCOON 2007*. LNCS, vol. 4598, pp. 75–85. Springer, Heidelberg (2007)
6. Fomin, F., Grandoni, F., Kratsch, D.: Measure and conquer: a simple $O(2^{0.288})$ independent set algorithm. In: *Proceedings of SODA*, pp. 18–25 (2006)
7. Frank, A.: Some polynomial algorithms for certain graphs and hypergraphs. In: *Proceedings of the Fifth British Combinatorial Conference*, pp. 211–226 (1975)
8. Gavril, F.: Maximum weight independent sets and cliques in intersection graphs of filaments. *Information Processing Letters* 73(5–6), 181–188 (2000)
9. Golumbic, M.: *Algorithmic Graph Theory and Perfect Graphs*. *Annals of Discrete Mathematics*, vol. 57. North-Holland Publishing Co., Amsterdam (2004)
10. Laskar, R., Shier, D.: On chordal graphs. *Congressus Numerantium* 29, 579–588 (1980)
11. Moran, S., Snir, S.: Convex recolorings of strings and trees: Definitions, hardness results and algorithms. *JCSS* 74(5), 850–869 (2008)
12. Ponta, O., Hüffner, F., Niedermeier, R.: Speeding up dynamic programming for some NP-hard graph recoloring problems. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) *TAMC 2008*. LNCS, vol. 4978, pp. 490–501. Springer, Heidelberg (2008)
13. Razgon, I.: A $2^{O(k)}$ poly(n) algorithm for the parameterized convex recoloring problem. *Information Processing Letters* 104(2), 53–58 (2007)

Strongly Chordal and Chordal Bipartite Graphs Are Sandwich Monotone*

Pinar Heggernes¹, Federico Mancini¹, Charis Papadopoulos², and R. Sritharan³

¹ Department of Informatics, University of Bergen, Norway
`pinar.heggernes@ii.uib.no`, `federico.mancini@ii.uib.no`

² Department of Mathematics, University of Ioannina, Greece
`charis@cs.uoi.gr`

³ Dept. of Computer Science, University of Dayton, USA
`srithara@notes.udayton.edu`

Abstract. A graph class is sandwich monotone if, for every pair of its graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ with $E_1 \subset E_2$, there is an ordering $e_1, \dots, e_k$ of the edges in $E_2 \setminus E_1$ such that $G = (V, E_1 \cup \{e_1, \dots, e_i\})$ belongs to the class for every i between 1 and k . In this paper we show that strongly chordal graphs and chordal bipartite graphs are sandwich monotone, answering an open question by Bakonyi and Bono from 1997. So far, very few classes have been proved to be sandwich monotone, and the most famous of these are chordal graphs. Sandwich monotonicity of a graph class implies that minimal completions of arbitrary graphs into that class can be recognized and computed in polynomial time. For minimal completions into strongly chordal or chordal bipartite graphs no polynomial-time algorithm has been known. With our results such algorithms follow for both classes. In addition, from our results it follows that all strongly chordal graphs and all chordal bipartite graphs with edge constraints can be listed efficiently.

1 Introduction

A graph class is *hereditary* if it is closed under induced subgraphs, and *monotone* if it is closed under subgraphs that are not necessarily induced. Every monotone graph class is also hereditary, since removing any edge keeps the graph in the class, but the converse is not true. For example perfect graphs are hereditary but not monotone, since we can create chordless odd cycles by removing edges. Some of the most well-studied graph properties are monotone [1,3] or hereditary [12]. Between hereditary and monotone graph classes are *sandwich monotone* graph classes. Monotonicity implies sandwich monotonicity (if we can remove any edge, we can also remove the edges in a particular order), which again implies being hereditary for graph classes that allow isolated vertices (if we can reach any subgraph in the class by removing edges, we can also reach induced subgraphs leaving the desired vertices isolated), but none of the reverse chain of

* This work is supported by the Research Council of Norway and National Security Agency, USA.

implications holds. In this paper we study hereditary graph classes that are not monotone, and we resolve the *sandwich monotonicity* of two of them: strongly chordal graphs and chordal bipartite graphs.

Chordal graphs are the most famous class of graphs that are sandwich monotone [23]. Besides this, split [13], chain [15], and threshold [15] graphs are the only known sandwich monotone classes. On the other hand we know that cographs, interval, proper interval, comparability, permutation, and trivially perfect graphs are *not* sandwich monotone [15]. The following graph classes have been candidates of sandwich monotonicity since an open question in 1997 [2]: strongly chordal, weakly chordal, and chordal bipartite.

Our main motivation for studying sandwich monotonicity comes from the problem of *completing* a given arbitrary graph into a graph class, meaning adding edges so that the resulting graph belongs to the desired class. For example, a *chordal completion* is a chordal supergraph on the same vertex set. A completion is *minimum* if it has the smallest possible number of added edges. The problem of computing minimum completions are applicable in several areas such as molecular biology, numerical algebra and, more generally, to areas involving graph modeling with some missing edges due to lacking data [11,19,22]. Unfortunately minimum completions into most interesting graph classes, including strongly chordal graphs [16,26], are NP-hard to compute [19]. However, minimum completions are a subset of minimal completions, and hence we can search for minimum among the set of minimal. A completion is *minimal* if no subset of the added edges can be removed from it without destroying the desired property.

If a graph class is sandwich monotone then a completion into the class is minimal if and only if *no single added edge* can be removed from it [13,15,23], making minimality of a completion much easier to check. For a graph class that can be recognized in polynomial time, sandwich monotonicity implies that minimal completions into this class can be computed in polynomial time. More importantly, it implies that whether a given completion is minimal can be decided in polynomial time, which is a more general problem. This latter problem is so far solvable for completions into only two non sandwich monotone classes: interval graphs [14] and cographs [18]. As an example of usefulness of a solution of this problem, various characterizations of minimal chordal completions [6,16] have made it possible to design approximation algorithms [19] and fast exact exponential time algorithms [10] for computing minimum chordal completions. A solution of this problem also allows the computation of minimal completions that are not far from minimum in practice [5]. With the results that we present in this paper, we are able to characterize minimal strongly chordal completions of arbitrary graphs and minimal chordal bipartite completions of arbitrary bipartite graphs.

In a COCOON 2008 paper, Kijima et al. give an efficient algorithm for the following problem [17]. Given two graphs on the same vertex set such that one is chordal and one is a subgraph of the other, list all chordal graphs that are sandwiched between the two graphs. In fact for the solution of this problem the only necessary property of chordal graphs is sandwich monotonicity. Hence, with

our results this problem can also be solved efficiently for strongly chordal and chordal bipartite graphs.

2 Preliminaries

We consider undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, we denote its vertex and edge set by $V(G) = V$ and $E(G) = E$, respectively, with $n = |V|$ and $m = |E|$.

The *neighborhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *closed neighborhood* of x is defined as $N_G[x] = N_G(x) \cup \{x\}$. If $S \subseteq V$, then the neighbors of S , denoted by $N_G(S)$, are given by $\bigcup_{x \in S} N_G(x) \setminus S$.

A *chord* of a cycle is an edge between two nonconsecutive vertices of the cycle. A chordless cycle on k vertices is denoted by C_k . A graph is *chordal* if it does not contain an induced C_k for $k > 3$. A *perfect elimination ordering* of a graph $G = (V, E)$ is an ordering $v_1, \dots, v_n$ of V such that for each i, j, k , if $i < j$, $i < k$, and $v_i v_j, v_i v_k \in E$ then $v_j v_k \in E$. Rose has shown that a graph is chordal if and only if it admits a perfect elimination ordering [21].

A vertex is called *simplicial* if the subgraph induced by its neighborhood is a clique. Observe that a perfect elimination ordering is equivalent to removing a simplicial vertex repeatedly until the graph becomes empty. For a vertex v , the *deficiency* $D(v)$ is the set of non-edges in $N(v)$; more precisely, $D(v) = \{xy \mid vx, vy \in E, xy \notin E\}$. Thus if $D(v) = \emptyset$, v is simplicial.

A *strong elimination ordering* of a graph $G = (V, E)$ is an ordering of the vertices $v_1, \dots, v_n$ of V such that for each i, j, k, l with $i \leq k$ and $i < l$, if $i < j$, $k < l$, $v_i v_k, v_i v_l \in E$ and $v_j v_k \in E$ then $v_j v_l \in E$. A graph is *strongly chordal* if it admits a strong elimination ordering. It is known that every induced subgraph of a strongly chordal graph is strongly chordal [8]. Moreover *every* strong elimination ordering is a perfect elimination ordering but the converse is not necessarily true. Thus all strongly chordal graphs are chordal.

Two vertices u and v of a graph G are called *compatible* if $N[u] \subseteq N[v]$ or $N[v] \subseteq N[u]$; otherwise they are called *incompatible*. Given two incompatible vertices u and v the *u -private neighbors* are exactly the vertices of the set $N[u] \setminus N[v]$. A vertex v of G is called *simple* if the neighbors of the vertices of $N[v]$ are linearly ordered by set inclusion, that is, the vertices of $N[v]$ are pairwise compatible. Clearly, any simple vertex is simplicial but not necessarily vice versa. An ordering $v_1, \dots, v_n$ of a graph G is called *simple elimination ordering* if for each $1 \leq i \leq n$, v_i is simple in the graph $G_i \equiv G[\{v_i, \dots, v_n\}]$.

Theorem 1 ([8]). *A graph is strongly chordal if and only if it has a simple elimination ordering.*

A *k -sun* (also known as *trampoline*), for $k \geq 3$, is the graph on $2k$ vertices obtained from a clique $\{c_1, \dots, c_k\}$ on k vertices and an independent set $\{s_1, \dots, s_k\}$ on k vertices and edges $s_i c_i, s_i c_{i+1}$, $1 \leq i < k$, and $s_k c_k, s_k c_1$.

Theorem 2 ([8]). *A chordal graph is strongly chordal if and only if it does not contain a k -sun as an induced subgraph.*

For our studies of strongly chordal graphs, we will need the following definitions regarding the neighborhood of a simple vertex x . We partition the sets $N(x)$ and $S(x) \equiv N(N(x)) \setminus \{x\}$. $(N_0, N_1, \dots, N_k)$ is a partition of $N(x)$ such that $N_0 = \{y \in N(x) \mid N[x] = N[y]\}$ and $N(N_0) \subset N(N_1) \cdots \subset N(N_k)$ where k is as large as possible.

These sets are also used to partition $S(x)$ into $(S_1, \dots, S_k)$ where $S_1 = N(N_1) \setminus N[x]$ and $S_i = N(N_i) \setminus (N(N_{i-1}) \cup N[x])$, for $2 \leq i \leq k$. We call the above partition a *simple partition* with respect to x .

In the context of a minimal completion of a given graph into a graph belonging to a given class, we say that a *completion* $G' = (V, E \cup F)$ of an arbitrary graph $G = (V, E)$ is any supergraph G' of G on the same vertex set with the property that G' belongs to the given graph class. If $\mathcal{C}$ is a graph class, then we refer to G' as a $\mathcal{C}$ completion of G . For instance, a strongly chordal completion of any graph $G = (V, E)$ is the complete graph on V . The edges that are in G' but not in G are called *added edges*. A $\mathcal{C}$ completion is *minimal* if no proper subset of the added edges, when added to the input graph G , results in a graph in the class.

Although sandwich monotonicity has been a well studied property since 1976 [23], it was first given a name and a proper definition in a COCOON 2007 paper [15]:

Definition 1 ([15]). *A graph class $\mathcal{C}$ is sandwich monotone if the following is true for any pair of graphs $G = (V, E)$ and $H = (V, E \cup F)$ in $\mathcal{C}$ with $E \cap F = \emptyset$: There is an ordering $f_1, f_2, \dots, f_{|F|}$ of the edges in F such that in the sequence of graphs $G = G_0, G_1, \dots, G_{|F|} = H$, where G_{i-1} is obtained by removing edge f_i from G_i , every graph belongs to $\mathcal{C}$.*

Observation 3 ([23,15]). *The following are equivalent on any graph class $\mathcal{C}$:*

- (i) $\mathcal{C}$ is sandwich monotone.
- (ii) *A $\mathcal{C}$ completion is minimal if and only if no single added edge can be removed without leaving $\mathcal{C}$.*

3 Strongly Chordal Graphs Are Sandwich Monotone

In this section we prove that strongly chordal graphs are sandwich monotone, and using this result we will characterize minimal strongly chordal completions of arbitrary graphs.

It is easy to see that if a single edge is added to a C_k with $k \geq 5$, then a $C_{k'}$ is created with $k' \geq 4$. First we show that a similar result holds for k -suns.

Observation 4. *For $k \geq 4$, if a single edge is added to a k -sun to produce a chordal graph, then a k' -sun with $k > k' \geq 3$ is created.*

Next, we show that when a new vertex is added to a strongly chordal graph ensuring that the new vertex is simple in the larger graph, the larger graph is also strongly chordal.

Lemma 1. *Let x be a simple vertex in a graph $G = (V, E)$. If $G - x$ is strongly chordal then G is strongly chordal.*

The following lemma is well-known for chordal graphs, and we want to show a similar result for strongly chordal graphs.

Lemma 2 ([23]). *Let G be a chordal graph and let x be a simplicial vertex of G . Removing an edge incident to x results in a chordal graph.*

Lemma 3. *Let G be a strongly chordal graph and let x be a simple vertex of G . Removing an edge incident to x results in a strongly chordal graph.*

Proof. Let xy be an edge incident to x . Since x is a simple vertex, there is a simple elimination ordering α for G that starts with x . Let i be the index for which y is simple in G_i , $1 \leq i \leq n$, with respect to α . Then α is a simple elimination ordering for $G - \{xy\}$ since x remains simple in $G - \{xy\}$ and y has the same neighborhood in the corresponding graph G_i .

Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two strongly chordal graphs such that $E \cap F = \emptyset$ and $F \neq \emptyset$. Let x be a simple vertex of G' and let $(N_0, N_1, \dots, N_k)$ and $(S_1, \dots, S_k)$ be a simple partition with respect to x . Let $u \in S_i$, $1 \leq i < k$. We denote by p_u the smallest index $i \leq p_u < k$ such that $v \in N_{p_u}$ and $vu \in E$. We define the following set of edges:

$$C(x) = \{uv \mid u \in S_i, v \in N_j, i \leq p_u < j \leq k\}.$$

Observe that $C(x)$ does not contain any edge of the form uv where $u \in S_i$ and $v \in N_i$, $1 \leq i \leq k$.

Lemma 4. *Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two strongly chordal graphs such that $E \cap F = \emptyset$ and $F \neq \emptyset$. There exists a simple vertex x of G' such that $F \not\subseteq D(x) \cup C(x)$.*

The following lemma describes that in a strongly chordal graph we can turn the neighborhood of any vertex into a clique by adding edges so that the strongly chordal property is preserved.

Lemma 5. *Let $G = (V, E)$ be a strongly chordal graph and let x be a vertex of G . Then $G' = (V, E \cup D(x))$ is a strongly chordal graph.*

Proof. Since G is strongly chordal it admits a simple elimination ordering. Let β be any simple elimination ordering of G . We prove that β is also a simple elimination ordering of G' . Observe first that β is a perfect elimination ordering of G' [23].

Assume for contradiction that β is not a simple elimination ordering of G' . Then there exist two adjacent vertices w_1, w_2 that are incompatible in G'_i , for some $1 \leq i \leq n$. This means that there exist at least two vertices z_1 and z_2 such that $w_1 z_1, w_2 z_2 \in E(G'_i)$ and $w_1 z_2, w_2 z_1 \notin E(G'_i)$. Since β is simple for G and we only add edges in G' , at least one of the edges $w_1 z_1, w_2 z_2$ is added because of x . If both of them are added because of x then all four vertices w_1, w_2, z_1, z_2 are adjacent to x in G and w_1, w_2 are compatible in G'_i .

Without loss of generality assume that $xw_1, xz_1 \in E(G)$ so that $w_1 z_1 \in E(G'_i)$. Now if $xw_2 \in E(G)$ or $xz_2 \in E(G)$ then $z_1 w_2 \in E(G'_i)$ or $z_2 w_1 \in E(G'_i)$,

respectively, meaning that w_1, w_2 are incompatible in G'_i . Thus $xw_2, xz_2 \notin E(G)$. Remember that by assumption we have $\beta^{-1}(w_1), \beta^{-1}(w_2), \beta^{-1}(z_1), \beta^{-1}(z_2) \geq i$. We consider now the position of x in the ordering β . If $\beta^{-1}(x) < i$ then β is not a perfect elimination ordering for G since $xw_1, xz_1 \in E(G)$, $\beta^{-1}(x) < \beta^{-1}(w_1)$ and $\beta^{-1}(x) < \beta^{-1}(z_1)$. Hence we are left with the case of $\beta^{-1}(x) \geq i$. But then in such a case β is not a simple elimination ordering for G_i since w_1, w_2 are incompatible in G_i because $w_1x, w_2z_2 \in E(G)$ and $w_2x, z_2x \notin E(G)$.

Therefore in all cases we get a contradiction and thus β is a simple elimination ordering of G' which implies that G' is strongly chordal.

Lemma 6. *Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two strongly chordal graphs such that $E \cap F = \emptyset$ and $F \neq \emptyset$. Let x be a simple vertex of G' such that $F \not\subseteq D(x) \cup C(x)$. The graph $H = (V, E \cup D(x) \cup C(x))$ is a strongly chordal graph.*

Now we are equipped with the necessary tools for proving the next important property of strongly chordal graphs.

Lemma 7. *Let $G = (V, E)$ and $G' = (V, E \cup F)$ be two strongly chordal graphs such that $E \cap F = \emptyset$ and $F \neq \emptyset$. Then there exists an edge $f \in F$ such that $G' - f$ is a strongly chordal graph.*

Proof. We prove the statement by induction on the number of vertices $|V|$. If $|V| \leq 3$ all graphs are strongly chordal and the statement holds. Assume that the statement is true for $|V| - 1$ vertices. Let x be a simple vertex of G' such that $F \not\subseteq D(x) \cup C(x)$. By Lemma 4 such a vertex always exist in G' . If there is an edge f of F incident to x then by Lemma 3 $G' - f$ is strongly chordal. Otherwise, no edge of F is incident to x and by Lemma 4 there is always an edge of F incident to a vertex of $V \setminus N_{G'}[x]$.

Now let $G_x = (V \setminus \{x\}, E \cup D(x) \cup C(x))$. G_x is strongly chordal by Lemma 6. Also the graph $G'_x = G'[V \setminus \{x\}]$ is strongly chordal as an induced subgraph of a strongly chordal graph. Notice that the added edges of G'_x are given by the set $F \setminus (D(x) \cup C(x))$ which by Lemma 4 is non-empty. Furthermore it is important to notice that G_x is a subgraph of G'_x since the set of edges $D(x)$ and $C(x)$ are edges of G' . Both graphs G_x and G'_x are on $|V| - 1$ vertices and by the induction hypothesis there is always an edge $f \in F \setminus (D(x) \cup C(x))$ such that $G'_x - f$ is strongly chordal.

Let $(N_0, N_1, \dots, N_k)$ and $(S_1, \dots, S_k)$ be a simple partition with respect to x in G' . Let us now show if the edge that is picked at the induction step is between vertices of $N(x)$ and $S(x)$ then there is an edge $f = uv \in F \setminus (D(x) \cup C(x))$ such that $u \in N_i$ and $v \in S_i$ and $G'_x - f$ is strongly chordal. If at the induction step uv is picked in such a way then we are done. Thus assume that $u \in N_j$ and $v \in S_i$ for $i < j$. We show first that there is an edge of the form $v'v \in F \setminus (D(x) \cup C(x))$ where $v' \in N_i$. By definition of $C(x)$, we know that $p_v \geq j > i$ because $uv \in F \setminus (D(x) \cup C(x))$. Then by the smallest choice of p_v , every edge incident to v and the vertices of $N_i, \dots, N_{p_v-1}$ belongs to $F \setminus (D(x) \cup C(x))$. Hence there is an edge of the form $v'v \in F \setminus (D(x) \cup C(x))$ where $v' \in N_i$. Now we show that the graph $G'_x - \{v'v\}$ is strongly chordal by using the fact that $G'_x - \{uv\}$ is strongly

chordal. Assume for the sake of contradiction that $G'_x - \{v'v\}$ is not strongly chordal. Since we remove only a single edge from a strongly chordal graph G'_x , there is a chordless cycle on four vertices or by Observation 4 there is a 3-sun in $G'_x - \{v'v\}$. If there is a chordless cycle in $G'_x - \{v'v\}$ then let v_a and v_b be the two non-adjacent vertices that are both adjacent to v' and v in G'_x . By the fact that $N_{G'_x}[v'] \subseteq N_{G'_x}[u]$ ($v' \in N_i$ and $u \in N_j$), u is adjacent to both vertices in G'_x and then we reach a contradiction to the chordal graph $G'_x - \{uv\}$. If there is a 3-sun in $G'_x - \{v'v\}$ then we consider two cases: (i) if v' belongs to the clique of the 3-sun then by $N_{G'_x}[v'] \subseteq N_{G'_x}[u]$ we know that u is also adjacent to the vertices of the clique of the 3-sun; thus we reach a contradiction to the strongly chordal graph $G'_x - \{uv\}$. (ii) If v' belongs to the independent set of the 3-sun then by $N_{G'_x}[v'] \subseteq N_{G'_x}[u]$, u is adjacent to the two vertices of the clique. If u is adjacent to at least one further vertex of the 3-sun then $G'_x - \{uv\}$ is not chordal. Otherwise the graph $G'_x - \{uv\}$ has a 3-sun and thus in both cases we reach a contradiction. Therefore if the edge f of the graph $G'_x - f$ is between vertices of $N(x)$ and $S(x)$ then there is an edge $f' = uv$ such that $f' \in F \setminus (D(x) \cup C(x))$, $u \in N_i$, $v \in S_i$ and the graph $G'_x - \{uv\}$ is strongly chordal.

Next we show that $G' - f$ is strongly chordal by constructing the graph according to $G'_x - f$. We prove that $G' - f$ is strongly chordal by using Lemma 1 and showing that x is a simple vertex in $G' - f$. If f is not between $N(x)$ and $S(x)$ then x is simple in $G' - f$ since x is simple in G' . Otherwise because of the previous result there is an edge $f \in F \setminus (D(x) \cup C(x))$ such that the endpoints of f belong to the sets N_i and S_i for some $1 \leq i \leq k$. Let u and v be the endpoints of f such that $u \in N_i$ and $v \in S_i$. Remember that in G' we have $N_{G'}(N_{i-1}) \subseteq N_{G'}(N_i)$. In the graph $G' - f$ we have that $N_{G'-f}(N_{i-1}) \subseteq N_{G'-f}(u) \subset N_{G'-f}(N_i \setminus \{u\})$ meaning that there is an inclusion set property for $N(x)$. Therefore x is a simple vertex of $G' - f$ and thus $G' - f$ is strongly chordal which completes the proof.

Theorem 5. *Strongly chordal graphs are sandwich monotone.*

By the previous theorem and Observation 3 we have the following property of minimal strongly chordal completions.

Corollary 1. *Let $G = (V, E)$ be an arbitrary graph and let $G' = (V, E \cup F)$ be a strongly chordal graph such that $E \cap F = \emptyset$ and $F \neq \emptyset$. G' is a minimal strongly chordal completion if and only if no edge of F can be removed from G' without destroying the strongly chordal property.*

It is known that an edge f can be removed from a chordal graph if and only if f is not the unique chord of a C_4 [23]. We prove a similar characterization for strongly chordal graphs. We call a *chord of a 3-sun* an edge between a vertex of the independent set and a vertex of the clique.

Lemma 8. *Let G be a strongly chordal graph and let f be an edge of G . $G - f$ is a strongly chordal graph if and only if f is not the unique chord of a C_4 or the unique chord of a 3-sun.*

The previous lemma leads to the following characterization of minimal strongly chordal completions.

Theorem 6. *Let $G = (V, E)$ be a graph and $G' = (V, E \cup F)$ be a strongly chordal completion of G . G' is a minimal strongly chordal completion if and only if every $f \in F$ is the unique chord of a C_4 or a 3-sun in G' .*

Proof. If G' is a minimal strongly chordal completion of G then $G' - f$ is not strongly chordal for any edge $f \in F$ by definition of minimality. Thus $G' - f$ contains either a chordless cycle or a chordless 3-sun as an induced subgraph by Lemma 8. If every edge $f \in F$ is the unique chord of a C_4 or a 3-sun in G' then $G' - f$ is not strongly chordal and by Corollary 1 G' is minimal.

4 Chordal Bipartite Graphs Are Sandwich Monotone

A bipartite graph $B = (X, Y, E)$ is *chordal bipartite* if it does not contain an induced C_k for $k \geq 6$. In this section we show that chordal bipartite graphs are sandwich monotone, answering an open question of Bakony and Bono [2]. Our approach is to make use of a well known relationship between the classes of strongly chordal graphs and chordal bipartite graphs and Lemma 7.

Theorem 7 ([7]). *Given a bipartite graph $B = (X, Y, E)$, let G be the graph obtained from B by adding edges between pairs of vertices in X so that X becomes a clique. Then, B is chordal bipartite if and only if G is strongly chordal.*

Lemma 9. *Let $B = (X, Y, E)$ and $B' = (X, Y, E \cup F)$ be chordal bipartite graphs with $E \cap F = \emptyset$ and $F \neq \emptyset$. Then, there exists an edge $f \in F$ such that $B' - f$ is chordal bipartite.*

Proof. Let $C = \{vw \mid v \in X, w \in X, v \neq w\}$. First construct the following graphs: $G = ((X \cup Y), (E \cup C))$ and $G' = ((X \cup Y), (E \cup C \cup F))$. By Theorem 7, G and G' are strongly chordal. By, Lemma 7, there exists $f \in F$ such that $G' - f$ is strongly chordal. The desired chordal bipartite graph $B' - f$ is obtained from $G' - f$, via Theorem 7, by simply deleting all the edges in C .

Hence the next theorem follows.

Theorem 8. *Chordal bipartite graphs are sandwich monotone.*

From the theorem above and Observation 3 we have the following corollary.

Corollary 2. *Let $B = (X, Y, E)$ be an arbitrary bipartite graph and let $B' = (X, Y, E \cup F)$ be a chordal bipartite graph such that $E \cap F = \emptyset$ and $F \neq \emptyset$. B' is a minimal chordal bipartite completion of B if and only if for any $f \in F$, $B' - f$ is not chordal bipartite.*

Lemma 10. *Let B be a chordal bipartite graph and f be an edge of B . $B - f$ is a chordal bipartite graph if and only if f is not the unique chord of a C_6 in B .*

Proof. If f is the unique chord of a C_6 in B , then $B - f$ contains a C_6 and hence is not chordal bipartite. For the other direction, observe that if the deletion of a single edge from a chordal bipartite graph creates an induced cycle on six or more vertices, then the created induced cycle must have exactly six vertices. Thus, if $B - f$ is not chordal bipartite, then f must be the unique chord of a C_6 in B .

Finally, we have the following characterization.

Theorem 9. *Let $B = (X, Y, E)$ be a bipartite graph and let $B' = (X, Y, E \cup F)$ be a chordal bipartite completion of B . B' is a minimal chordal bipartite completion if and only if every $f \in F$ is the unique chord of a C_6 in B' .*

5 Concluding Remarks

We have proved that strongly chordal graph and chordal bipartite graphs are sandwich monotone. The best running time for recognizing those graphs is $\mathcal{O}(\min\{m \log n, n^2\})$ [20,25]. Hence by applying a simple algorithm proposed in [15] we obtain algorithms for computing minimal completions into both graph classes of arbitrary graphs with running time $\mathcal{O}(n^4(\min\{m \log n, n^2\}))$. We strongly believe that such a running time can be improved. Furthermore problems that involve listing all strongly chordal graphs (or chordal bipartite graphs) between a given pair of graphs where at least one of the input pair is strongly chordal (or chordal bipartite) can be efficiently solved by generalizing the results in [17].

A graph is weakly chordal if neither the graph nor its complement contains a chordless cycle longer than 4. Minimum *weakly chordal* completions are NP-hard to compute [4], and we do not yet know whether minimal weakly chordal completions can be computed or recognized in polynomial time. We would like to know whether weakly chordal graphs are sandwich monotone. The resolution of this question in the affirmative would answer the above questions about minimal weakly chordal completions. Another interesting question to resolve is whether minimum chordal bipartite completions are NP-hard to compute. This is widely believed, but no proof of it exists to our knowledge. Note that the related sandwich problem was solved only quite recently [9,24].

References

1. Alon, N., Shapira, A.: Every monotone graph property is testable. In: STOC 2005, pp. 128–137 (2005)
2. Bakonyi, M., Bono, A.: Several results on chordal bipartite graphs. Czechoslovak Math. J. 46, 577–583 (1997)
3. Balogh, J., Bolobás, B., Weinreich, D.: Measures on monotone properties of graphs. Disc. Appl. Math. 116, 17–36 (2002)
4. Burzyn, P., Bonomo, F., Duran, G.: NP-completeness results for edge modification problems. Discrete Applied Math. 99, 367–400 (2000)
5. Bodlaender, H.L., Koster, A.M.C.A.: Safe separators for treewidth. Discrete Math. 306, 337–350 (2006)
6. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. SIAM J. Comput. 31, 212–232 (2001)
7. Dahlhaus, E.: Chordale graphen im besonderen hinblick auf parallele algorithmen, Habilitation thesis, Universität Bonn (1991)
8. Farber, M.: Characterizations on strongly chordal graphs. Discrete Mathematics 43, 173–189 (1983)

9. de Figueiredo, C.M.H., Faria, L., Klein, S., Sritharan, R.: On the complexity of the sandwich problems for strongly chordal graphs and chordal bipartite graphs. *Theoretical Computer Science* 381, 57–67 (2007)
10. Fomin, F.V., Kratsch, D., Todinca, I., Villanger, Y.: Exact algorithms for treewidth and minimum fill-in. *SIAM J. Computing* 38, 1058–1079 (2008)
11. Goldberg, P.W., Golumbic, M.C., Kaplan, H., Shamir, R.: Four strikes against physical mapping of DNA. *J. Comput. Bio.* 2(1), 139–152 (1995)
12. Golumbic, M.C.: *Algorithmic Graph Theory and Perfect Graphs*, 2nd edn. *Annals of Discrete Mathematics*, vol. 57. Elsevier, Amsterdam (2004)
13. Heggenes, P., Mancini, F.: Minimal split completions. *Discrete Applied Mathematics* (in print); also In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) *LATIN 2006*. LNCS, vol. 3887, pp. 592–604. Springer, Heidelberg (2006)
14. Heggenes, P., Suchan, K., Todinca, I., Villanger, Y.: Characterizing minimal interval completions: Towards better understanding of profile and pathwidth. In: Thomas, W., Weil, P. (eds.) *STACS 2007*. LNCS, vol. 4393, pp. 236–247. Springer, Heidelberg (2007)
15. Heggenes, P., Papadopoulos, C.: Single-Edge Monotonic Sequences of Graphs and Linear-Time Algorithms for Minimal Completions and Deletions. *Theoretical Computer Science* 410, 1–15 (2009); also In: Lin, G. (ed.) *COCOON 2007*. LNCS, vol. 4598, pp. 406–416. Springer, Heidelberg (2007)
16. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.* 28, 1906–1922 (1999)
17. Kijima, S., Kiyomi, M., Okamoto, Y., Uno, T.: On listing, sampling, and counting the chordal graphs with edge constraints. In: Hu, X., Wang, J. (eds.) *COCOON 2008*. LNCS, vol. 5092, pp. 458–467. Springer, Heidelberg (2008)
18. Lokshtanov, D., Mancini, F., Papadopoulos, C.: Characterizing and Computing Minimal Cograph Completions. In: Preparata, F.P., Wu, X., Yin, J. (eds.) *FAW 2008*. LNCS, vol. 5059, pp. 147–158. Springer, Heidelberg (2008)
19. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. *Disc. Appl. Math.* 113, 109–128 (2001)
20. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. *SIAM J. Comput.* 16, 973–989 (1987)
21. Rose, D.: Triangulated graphs and the elimination process. *J. Math. Anal. Appl.* 32, 597–609 (1970)
22. Rose, D.J.: A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: Read, R.C. (ed.) *Graph Theory and Computing*, pp. 183–217. Academic Press, New York (1972)
23. Rose, D., Tarjan, R.E., Lueker, G.: Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.* 5, 266–283 (1976)
24. Sritharan, R.: Chordal bipartite completion of colored graphs. *Discrete Mathematics* 308, 2581–2588 (2008)
25. Spinrad, J.P.: Doubly lexical ordering of dense 0-1 matrices. *Information Processing Letters* 45, 229–235 (1993)
26. Yannakakis, M.: Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.* 2, 77–79 (1981)

Hierarchies and Characterizations of Stateless Multicounter Machines^{*}

Oscar H. Ibarra and Ömer Egecioğlu

Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
{ibarra,omer}@cs.ucsb.edu

Abstract. We investigate the computing power of stateless multicounter machines with reversal-bounded counters. Such a machine can be deterministic, nondeterministic, realtime (the input head moves right at every step), or non-realtime. The deterministic realtime stateless multicounter machines has been studied elsewhere [1]. Here we investigate non-realtime machines in both deterministic and nondeterministic cases with respect to the number of counters and reversals. We also consider closure properties and relate the models to stateless multihead automata and show that the bounded languages accepted correspond exactly to semilinear sets.

Keywords: Stateless multicounter machine, reversal-bounded, non-realtime, hierarchy, stateless multihead automata, semilinear set, closure properties.

1 Introduction

Stateless machines (i.e. machines with only one state) have been investigated in recent papers because of their connection to certain aspects of membrane computing [9], a subarea of molecular computing that was introduced in a seminal paper by Gheorge Păun [7] (see also [8]).

Stateless machines have no states to store information. The move of such a machine depends only on the symbol(s) scanned by the input head(s) and the local portion of the memory unit(s). Acceptance of an input string has to be defined in a different way. For example, in the case of a pushdown automaton (PDA), one definition of acceptance is by “null” stack. It is well known that nondeterministic PDA with states are equivalent to stateless nondeterministic PDA [2] although this is not true for the deterministic case [5]. For Turing Machines where acceptance is when the machine enters a halting configuration, it can be shown that the stateless version is less powerful than those with states. In [4,9] the computing power of stateless multihead automata with respect to decision problems and head hierarchies were investigated. For these devices, the input is provided with left and right end markers. The move depends only on the

^{*} This research was supported in part by NSF Grants CCF-0430945 and CCF-0524136.

symbols scanned by the input heads. The machine can be deterministic, nondeterministic, one-way, two-way. An input is accepted if, when all heads are started on the left end marker, the machine eventually reaches the configuration where all heads are on the right end marker. In [6], various types of stateless restarting automata and two-pushdown automata were compared to the corresponding machines with states.

We investigate the computing power of stateless multicounter machines with reversal-bounded counters. Such a machine operates on a one-way input delimited by left and right end markers, and it is equipped with m counters. A move depends only on the symbol under the input head and the signs of the counters (zero or positive). An input string is accepted if, when the input head is started on the left end marker with all counters zero, the machine eventually reaches the configuration where the input head is on the right end marker with all the counters again zero. Moreover, in the computation, no counter makes more than k -reversals (alternation between increasing mode and decreasing mode and vice-versa) for a specified k . There are various scenarios to consider: deterministic, nondeterministic, realtime (the input head moves right at every step), non-realtime. In this paper we investigate mainly the non-realtime deterministic and nondeterministic machines.

2 Stateless Multicounter Machines

A deterministic stateless (one-way) m -counter machine operates on an input of the form $\epsilon w \$$, where ϵ and $\$$ are the left and right end markers for the input w . At the start of the computation, the input head is on the left end marker ϵ and all m counters are zero. The moves of the machine are described by a set of rules of the form:

$$(x, s_1, \dots, s_m) \rightarrow (d, e_1, \dots, e_m)$$

where $x \in \Sigma \cup \{\epsilon, \$\}$, Σ is the input alphabet, $s_i = \text{sign of counter } C_i$ (0 or 1 for positive), $d = 0$ or 1 (direction of the move of the input head: $d = 0$ means don't move, $d = 1$ means move the head one cell to the right), and $e_i = +, -, \text{ or } 0$ (increment counter i by 1, decrement counter i by 1, or do not change counter i), with the restriction that $e_i = -$ is applicable only if $s_i = 1$. For a deterministic machine, no two rules have the same left hand sides.

The input w is accepted if the machine reaches the configuration where the input head is on the right end marker $\$$ and all counters are zero. The machine is k -reversal if it has the property that each counter makes at most k "full" alternations between increasing mode and decreasing mode and vice-versa on any computation (accepting or not). Thus, e.g., $k = 2$ means the counter can only go from increasing to decreasing to increasing to decreasing. A machine is reversal-bounded if it is k -reversal for some k .

Note that when the input head reaches $\$$, the machine can continue computing until all counters eventually become zero for an input that is accepted. A special case is when the machine is *realtime*: in this case $d = 1$ for each rule, i.e., the input

head moves right at each step. This means that when the input head reaches \$, all the counters must be zero for the input to be accepted. Deterministic realtime machines were investigated in [1], where hierarchies with respect to the number of counters and number of reversals were studied. A stateless multicounter machine is nondeterministic if different rules are allowed to have identical left hand sides.

2.1 1-Reversal Machines

For $w \in \Sigma^*$ and $a \in \Sigma$, define $|w|_a$ as the number of occurrences of a in w . Clearly, any language accepted by a realtime machine can be accepted by a non-realtime machine. The latter is strictly more powerful.

Theorem 1. *The language $L = \{w \mid w \in \{a, b\}^*, |w|_a = |w|_b\}$ can be accepted by a stateless non-realtime 1-reversal 2-counter machine M but not by a stateless realtime k -reversal m -counter machine for any $k, m \geq 1$.*

Proof. M has counters C_1 and C_2 . On input $\epsilon w \$$, M reads the input and stores the number of a 's (resp., b 's) it sees in C_1 (resp., C_2). When the input head reaches \$, the counters are decremented simultaneously while the head remains on \$. M accepts if and only if the counters become zero at the same time.

Suppose L is accepted by some realtime k -reversal m -counter machine M' . Let x be a string with $|x|_a = |x|_b > 0$. Then x is accepted by M' , i.e., M' on input $\epsilon x \$$, starts with the input head on ϵ with all counters zero, computes, and accepts after reading the last symbol of x with all counters again at zero.

Consider now giving input xab to M' . After processing x , all counters are zero. Clearly, after processing symbol a , at least one counter of M' must increment; otherwise (i.e., if all counters remain at zero), M' will accept all strings of the form xa^i for all i , a contradiction. Then after processing b , all counters must again be zero, since xab is in L . It follows that on input xab , at least one counter made an additional reversal than on input x . Repeating the argument, we see that for some i , $x(ab)^i$ will require at least one counter to make $k + 1$ reversals. Therefore M' cannot be k -reversal for any k . $\square$

The result above can be made stronger. A non-realtime reversal-bounded multicounter machine is *restricted* if it can only accept an input when the input head first reaches the right end marker \$ and all counters are zero. Hence, there is no applicable rule when the input head is on \$. However, the machine can be non-realtime (i.e., need not move at each step) when the head is not on \$. The machine can also be nondeterministic. An argument similar to the proof of Theorem 1 can be used to prove the following result:

Corollary 1. *$L = \{w \mid w \in \{a, b\}^*, |w|_a = |w|_b\}$ cannot be accepted by any stateless restricted nondeterministic non-realtime reversal-bounded multicounter machine.*

Next, we give an example of a unary singleton language that is accepted by a non-realtime 1-reversal machine that does not seem to be acceptable by a realtime 1-reversal machine. Our construction uses a technique in [4] where it was shown that the language can be accepted by a stateless $(2m + 1)$ -head machine.

Theorem 2. *For every $m \geq 1$, the singleton language $L = \{a^{2^m-1}\}$ can be accepted by a stateless non-realtime 1-reversal $2(m + 1)$ -counter machine M .*

Proof. Let the counters be $0, 1, \dots, 2m + 1$. Counters $2, \dots, 2m + 1$ form m pairs $(i, i + m)$. Initially all counters are zero and the input head is on the left end marker. We describe the computation of M on input $\phi a^n \$$ in two phases.

Loading phase:

In this phase, the input head is moved to the right while simultaneously counters $1, \dots, 2m + 1$ are incremented by 1 for every right move of the head. When the input head reaches $\$$, counter 0 has value zero and counters $1, \dots, 2m + 1$ have value $n + 1$. Then M enters the next phase.

Computing phase:

When this phase is entered, the input head is on the right end marker, counter 0 has value zero and counters $1, \dots, 2m + 1$ each have value $n + 1$. We will refer to counter 1 as the head counter. The input head remains on the right end marker during this phase.

First, counter 0 is incremented by 1, counters $1, 2, \dots, m + 1$ (i.e., the main counter and the first counter from each pair) are decremented by 1, while counter $m + 2, \dots, 2m + 1$ (second components of all pairs) remain unchanged with value $n + 1$.

Then counters $(m + 1, 2m + 1)$ (that is, the last pair), whose difference (in value) is one, are decremented until $m + 1$ becomes zero. From here, counters $1, 2, \dots, m$ (the main counter and the first components of all unused pairs) are decremented simultaneously with counter $2m + 1$, until counter $2m + 1$ becomes zero. This will take only one step, and after that counters $1, 2, \dots, m$ will have value $n - 1$, counters $m + 2, \dots, 2m$ will have value $n + 1$, while counters $m + 1$ and $2m + 1$ will have value zero.

Then the next pair $(m, 2m)$ is taken, and the same sequence of steps is repeated. Note that the difference in values between these counters is now 2. The result is that counters m and $2m$ are decremented to zero, while counters $1, 2, \dots, m - 1$ will have value $n - 3$. This is continued with the rest of the pairs, until the following configuration is reached: counter 0 has value 1, counters 1 and 2 have values 2^{m-1} , counter $m + 2$ has value $n + 1$, and all other counters are zero.

From here, counters 2 and $m + 2$ are decremented until counter 2 becomes zero. At this point, counters 1 and $m + 2$ have same value if and only if the length of the string is $2^m - 1$. After that counters 1 and $m + 2$ are decremented and the input is accepted if and only if these counters become zero at the same time. This happens if and only if the input has length $2^m - 1$. $\square$

2.2 k -Reversal Machines

We will be using the following result (shown in [1]) relating the number of reversals to counters.

Theorem 3. *If a language L is accepted by a stateless non-realtime k -reversal m -counter machine then it can be accepted by a stateless non-realtime 1-reversal $(2k - 1)m$ -counter machine.*

2.3 Counter and Reversal Hierarchies

First we prove that there is a hierarchy with respect to the number of counters for stateless non-realtime machines.

Lemma 1. *For $k, m \geq 1$, there is a unique maximal number $f(k, m)$ such that the singleton language $L = \{a^{f(k, m)}\}$ is accepted by a stateless non-realtime k -reversal m -counter machine. (We refer to L as “maximal”.)*

Proof. Follows from the fact that the singleton language $\{a\}$ is accepted by a non-realtime 1-reversal 1-counter machine and the fact that the number of non-realtime k -reversal m -counter machines is finite, depending only on k and m . $\square$

Theorem 4. *For $m \geq 1$, $m + 1$ counters can do more than m counters for stateless non-realtime k -reversal machines.*

Proof. Clearly, any language accepted by a non-realtime k -reversal m -counter machine can be accepted by a k -reversal $(m + 1)$ -counter machine.

Now let M a non-realtime k -reversal m -counter machine accepting the maximal language $\{a^n\}$ (for some n). Such a language exists by the above lemma. Let the counters of M be $C_1, \dots, C_m$. We will construct a non-realtime k -reversal $(m + 1)$ -counter machine M' accepting $\{a^{n+1}\}$. It would then follow that $m + 1$ counters are better than m counters.

M' will have counters $C_1, \dots, C_m, C_{m+1}$ and its rules are defined as follows:

1. If $(\$, s_1, \dots, s_m) \rightarrow (0, e_1, \dots, e_m)$ is in M , then $(\$, s_1, \dots, s_m, 0) \rightarrow (0, e_1, \dots, e_m, 0)$ is in M' .
2. If $(\$, s_1, \dots, s_m) \rightarrow (1, e_1, \dots, e_m)$ is in M , then $(\$, s_1, \dots, s_m, 0) \rightarrow (1, e_1, \dots, e_m, 1)$ is in M' .
3. If $(a, s_1, \dots, s_m) \rightarrow (d, e_1, \dots, e_m)$ is in M , then $(a, s_1, \dots, s_m, 1) \rightarrow (1, 0, \dots, 0, -1)$ is in M' .
4. If $(x, s_1, \dots, s_m) \rightarrow (d, e_1, \dots, e_m)$ is in M , then $(x, s_1, \dots, s_m, 0) \rightarrow (d, e_1, \dots, e_m, 0)$ is in M' for $x \in \{a, \$\}$. $\square$

Note that the above result and proof hold for the realtime case. In fact, in the construction, case 1 does not apply; in case 2, $s_1 = \dots = s_m = 0$, and in case 4, $x = a$ and $d = 1$. The following result can be shown:

Theorem 5. *For any fixed m and $k < 2^{\frac{m}{2}-1}/m$, there is a language accepted by a stateless $(k + 1)$ -reversal m -counter machine which is not accepted by any stateless k -reversal m -counter machine.*

2.4 Closure Properties

Theorem 6. *The class of languages accepted by stateless deterministic non-realtime k -reversal multicounter machines is closed under intersection, union, and complementation.*

Proof. Let M_1 and M_2 be two such machines.

Intersection:

Let M_1 and M_2 have m and n counters, respectively. We construct a machine M which simulates these machines in parallel. M has $m + n$ counters to simulate the counters of M_1 and M_2 , using the following rules:

1. If $(x, s_1, \dots, s_m) \rightarrow (d, e_1, \dots, e_m)$ in M_1 and $(x, s'_1, \dots, s'_n) \rightarrow (d, e'_1, \dots, e'_n)$ in M_2 , then $(x, s_1, \dots, s_m, s'_1, \dots, s'_n) \rightarrow (x, e_1, \dots, e_m, e'_1, \dots, e'_n)$ in M .
2. If $(x, s_1, \dots, s_m) \rightarrow (0, e_1, \dots, e_m)$ in M_1 and $(x, s'_1, \dots, s'_n) \rightarrow (1, e'_1, \dots, e'_n)$ in M_2 , then $(x, s_1, \dots, s_m, s'_1, \dots, s'_n) \rightarrow (x, e_1, \dots, e_m, 0, \dots, 0)$ in M .
3. If $(x, s_1, \dots, s_m) \rightarrow (1, e_1, \dots, e_m)$ in M_1 and $(x, s'_1, \dots, s'_n) \rightarrow (0, e'_1, \dots, e'_n)$ in M_2 , then $(x, s_1, \dots, s_m, s'_1, \dots, s'_n) \rightarrow (x, 0, \dots, 0, e'_1, \dots, e'_n)$ in M .

Complementation and Union:

Given M_1 , we construct a machine M which accepts the complement of the language accepted by M_1 . In the addition to the m counters of M_1 , M uses a new counter T . Before the simulation, M sets T to 1. Then M simulates M_1 . If M_1 does not accept the input, either by getting stuck at some point on the input or reaching $\$$ and not able to zero all the counters, M decrements all the counters to zero and sets T to 0. Closure under union follows, since the class of languages is closed under intersection. $\square$

2.5 A Pumping Lemma for Stateless Deterministic Non-realtime Reversal-Bounded Multicounter Machines

We have the following “pumping lemma” type result for stateless deterministic non-realtime reversal-bounded multicounter machines.

Theorem 7. *Suppose L is the language accepted by a stateless deterministic non-realtime reversal-bounded m -counter machine over a unary alphabet. If L is infinite, then there exists some $n_0 \geq 0$ such that for $n \geq n_0$, $a^n \in L$ implies $a^{n+1} \in L$.*

Proof. We omit the proof.

As a corollary of Theorem 7, we have

Corollary 2. *The language $L = \{a^{2^n} \mid n \geq 1\}$ cannot be accepted by any stateless deterministic non-realtime reversal-bounded multicounter machine.*

3 Equivalence to Stateless Multihead Automata

It turns out that stateless non-realtime reversal-bounded multicounter machines over a unary alphabet are equivalent to stateless multihead automata.

A stateless m -head machine (over unary input) M operates on an input $\$a^n\$$, where $\$$ and $\$$ are the left and right end markers. The initial configuration is when all m heads are on the left end marker $\$$, and the accepting configuration is when all m heads reach the right end marker $\$$, which we assume is a halting configuration. The moves are defined by a set of rules of the form:

$$(\ell_1, \dots, \ell_m) \rightarrow (d_1, \dots, d_m)$$

where ℓ_i is the symbol under head i (can be $\$, a, \$$), $d_i = 0$ or 1 (direction of move of head i : no move or move right one cell). Note that since the machine is deterministic, no two rules can have the same left hand sides. Also there is no rule with left hand side $(\$, \dots, \$)$.

Lemma 2. *Any stateless multihead automaton M can be simulated by a stateless non-realtime 1-reversal multicounter machine M' .*

Proof. Let the heads of M be $H_1, \dots, H_m$. M' will have an input head and $2m$ counters $C_1, \dots, C_m, T_1, \dots, T_m$, which are initially zero. When given $\$a^n\$$, M' reads the input while simultaneously incrementing the m counters $C_1, \dots, C_m$. When the input head reaches the right end marker $\$$, each C_i will have value $n + 1$. Then M' simulates M . The input head of M' remains on $\$$ during the simulation. Counter C_i simulates the actions of head H_i . Moving head H_i one cell to the right right is simulated by decrementing C_i . Note that at the start of the simulation, $T_1, \dots, T_m$ are zero. This corresponds to the configuration when all the heads of M are on the left end marker $\$$. When C_i is first decremented (corresponding to H_i moving right of $\$$), T_i is set to 1. When the counters $C_1, \dots, C_m$ become zero (corresponding to all heads $H_1, \dots, H_m$ reaching $\$$), the T_i 's are decremented and the input is accepted. $\square$

Lemma 3. *Any stateless non-realtime 1-reversal multicounter machine can be simulated by a stateless multihead automaton M' .*

Proof. (Sketch.) First consider a stateless non-realtime 1-reversal 1-counter machine M with input head H and counter C . We construct a stateless multihead automaton M' equivalent to M . M' will have 6 heads $H_1, H_2, C_1, C_2, T_1, T_2$. Initially, all heads of M' are on $\$$. M' simulates M as follows:

1. Heads H_1 and C_1 simulate head H and counter C of M , respectively, where “incrementing” C corresponds to “moving” C_1 to the right on the input.
2. When C decrements, M' moves T_1 right to the next symbol (indicating a new situation).
3. M' restarts the simulation of M (from the beginning) but now using H_2 and C_2 (to simulate H and C) and at same time C_1 is moved along with C_2 when the latter is incrementing.

4. When C_2 decrements, M' moves T_2 right to the next symbol (indicating yet again another situation) and suspend the simulation, i.e., H_2 and C_2 are not moved. At this time, if C_2 is in position d , then C_1 is in position $2d$, i.e., the “distance” between these heads is d .
5. C_1 and C_2 are moved to the right in parallel until C_1 reaches $\$$. Note that the “distance” between C_1 (which is now on $\$$) and C_2 is still d .
6. Then M' uses H_2 and C_2 to resume the simulation of M , but C_2 now simulates the decreasing phase of counter C of M by moving right on the input. C_2 reaching $\$$ indicates that counter C of M has value 0.

M' accepts the language accepted by M . When M has several 1-reversal counters, the construction of M' above can be generalized. We omit the details. $\square$

From Theorem 3 and the above lemmas, we have the following characterization:

Theorem 8. *A language L over a unary alphabet is accepted by a stateless non-realtime reversal-bounded multicounter machine if and only if it can be accepted by a stateless multihead automaton.*

4 Unbounded Reversal Multicounter Machines

First we give an example of a stateless non-realtime counter machine that accepts the language $L = \{a^{2^i} \mid i \geq 0\}$. What is interesting is that this can be accepted by a machine with only 4 counters.

Here the input is $\$a_1a_2 \cdots a_n\$$ with the read head initially on the left end marker $\$$ and all m counters zero. The head moves to the right at each step. Depending on the symbol under the head and the signs of the counters, a counter is decremented (if positive), incremented, or left the same. Once the head reaches the $\$$ sign, further moves are possible, depending on the signs of the counters only. The machine accepts if the counters all become zero. Since further moves are allowed after the head reaches the $\$$, the machine is non-realtime.

Let us consider the unary alphabet. The inputs are of the form $\$a^n\$$. We can show the following:

Proposition 1. *The language $L = \{a^{2^i} \mid i \geq 0\}$ is accepted by a stateless non-realtime 4-counter machine.*

It can also be proved that by adding a fifth counter, the construction for the proof of the above proposition can be modified to accept the language

$$L = \{a^n \mid n \text{ is a tower of } 2\text{'s}\}.$$

Furthermore the singleton language $L = \{a^n \mid n = m \text{ levels of } 2\text{'s}\}$ can be accepted by a machine with $\log m + 5$ counters. In these examples, the counter machines are non-realtime. Interestingly, one can show that similar results can be obtained for realtime machines.

5 Nondeterministic Machines and Semilinear Sets

Recall that in a nondeterministic machine some rules can have the same left hand sides. In this section, we characterize bounded languages accepted by stateless nondeterministic reversal-bounded non-realtime multicounter machines in terms of semilinear sets.

In contrast to the fact that $L = \{a^{2n} \mid n \geq 0\}$ cannot be accepted by any stateless deterministic non-realtime reversal-bounded multicounter machine (Theorem 2), we can show that $L = \{a^n \mid n \geq 0\}$ is accepted by a stateless nondeterministic non-realtime 1-reversal multicounter machine if and only if L is regular. In fact, we can prove a more general result.

A language L is *bounded* if there are distinct symbols $a_1, \dots, a_r$ such that $L \subseteq a_1^* \cdots a_r^*$. The Parikh map of L , $\psi(L)$, is defined to be the set of r -tuples of nonnegative integers $\{(i_1, \dots, i_r) \mid a^{i_1} \cdots a^{i_r} \in L\}$.

Let $\mathbb{N}$ be the set of nonnegative integers and r be a positive integer. A subset Q of $\mathbb{N}^r$ is a *linear set* if there exist vectors $v_0, v_1, \dots, v_t$ in $\mathbb{N}^r$ such that $Q = \{v \mid v = v_0 + a_1 v_1 + \cdots + a_t v_t, a_i \in \mathbb{N}\}$. A set $Q \subseteq \mathbb{N}^r$ is *semilinear* if it is a finite union of linear sets.

It is known that $L \subseteq a_1^* \cdots a_r^*$ is accepted by a nondeterministic non-realtime reversal-bounded multicounter machine with states if and only if $\psi(L)$ is semilinear. This result also holds for stateless machines. We illustrate with an example below.

Consider the linear set $Q = \{(2, 1) + x(2, 3) + y(1, 0) \mid x, y \geq 0\}$. The bounded language corresponding to this set is $L = \{a^{2x+y+2}b^{3x+1} \mid x, y \geq 0\}$. We will construct a stateless nondeterministic non-realtime 1-reversal multicounter machine M accepting L . In the construction, we use some special types of counters, which we call *switches*. A switch starts at zero at the beginning of the computation (when the input head is on ϵ), then it is incremented to 1 at some point during the computation, and finally set back to zero before acceptance.

M has counters A_1, A_2, B_1, B_2, B_3 and other counters used as switches. Given input $\epsilon w \$$, we may assume that $w = a^m b^n$ for some m, n ; otherwise, we can use a switch counter to confirm that a b cannot be followed by an a .

On input $\epsilon a^m b^n \$$, M operates as follows: while the input is on ϵ , M increments A_1, A_2, B_1, B_2, B_3 simultaneously, x times, where x is chosen nondeterministically, after which (using some switches) increments counter A_3 , y times, where y is chosen nondeterministically. Then it checks that $m = A_1 + A_2 + A_3 + 2$. This is done by first reading 2 a 's (again using some switches). Then it reads the rest of the a -segment while decrementing A_1 until it becomes zero, then decrementing A_2 until it too becomes zero, and decrementing A_3 until it becomes zero. M 's head will be on the first b if and only if $m = A_1 + A_2 + A_3 + 2$. Similarly, by reading the b -segment, M can check that $m = B_1 + B_2 + B_3 + 1$, and this holds if and only if the head reaches $\$$ when counter B_3 becomes zero.

If Q is a semilinear set we can construct a machine for each linear set and then combine these machines into one machine that nondeterministically selects one of the machines to simulate. (We will need to use additional switches for this.)

One can formalize the discussion above to prove the “if” part of the next result. The “only if” part follows from the fact that it holds for machines with states [3].

Theorem 9. *$L \subseteq a_1^* \cdots a_r^*$ can be accepted by a stateless nondeterministic non-realtime reversal-bounded multicounter machine if and only if $\psi(L)$ is semilinear.*

Corollary 3. *$L \subseteq a_1^* \cdots a_r^*$ is accepted by a stateless nondeterministic non-realtime reversal-bounded multicounter machine with states if and only if it can be accepted by a stateless nondeterministic non-realtime reversal-bounded multicounter machine.*

References

1. Eğecioğlu, Ö., Ibarra, O.H.: On stateless multicounter machines. TR2009-01, Department of Computer Science, UCSB. In: Proceedings of Computability in Europe 2009 (CiE), Heidelberg, Germany (2009)
2. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Series in Computer Science. Addison-Wesley, Reading (1979)
3. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. Assoc. for Computing Machinery 25, 116–133 (1978)
4. Ibarra, O.H., Karhumäki, J., Okhotin, A.: On stateless multihead automata: Hierarchies and the emptiness problem. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 94–105. Springer, Heidelberg (2008)
5. Korenjak, A.J., Hopcroft, J.E.: Simple deterministic languages. In: Proceedings of IEEE 7th Annu. Symp. on Switching and Automata Theory, pp. 36–46 (1966)
6. Kutrib, M., Messerschmidt, H., Otto, F.: On stateless two-pushdown automata and restarting automata. In: Pre-Proceedings of the 8th Automata and Formal Languages (May 2008)
7. Păun, G.: Computing with Membranes. Journal of Computer and System Sciences 61(1), 108–143 (2000); Turku Center for Computer Science-TUCS Report 208 (November 1998), www.tucs.fi
8. Păun, G.: Computing with Membranes: An Introduction. Springer, Berlin (2002)
9. Yang, L., Dang, Z., Ibarra, O.H.: On stateless automata and P systems. In: Pre-Proceedings of Workshop on Automata for Cellular and Molecular Computing (August 2007)

Efficient Universal Quantum Circuits

Debajyoti Bera^{1,*}, Stephen Fenner^{2,**}, Frederic Green^{3,***},
and Steve Homer^{1,*}

¹ Boston University, Department of Computer Science, Boston, MA 02134
`{dbera,homer}@cs.bu.edu`

² University of South Carolina, Department of Computer Science and Engineering,
Columbia, SC 29208
`fenner@cse.sc.edu`

³ Clark University, Department of Mathematics and Computer Science, Worcester,
MA 01610
`fgreen@black.clarku.edu`

Abstract. We define and construct efficient depth-universal and almost-size-universal quantum circuits. Such circuits can be viewed as general-purpose simulators for central quantum circuit classes and used to capture the computational power of the simulated class. For depth we construct universal circuits whose depth is the same order as the circuits being simulated. For size, there is a log factor blow-up in the universal circuits constructed here which is nearly optimal for polynomial size circuits.

1 Introduction

Quantum computing is most naturally formulated using the quantum circuit model [Y93]. Many quantum algorithms are expressed in terms of (uniform) circuits which depend strongly on the problem being solved. However, the notion of a universal quantum computer is more naturally captured by quantum Turing machines [D85]. This being the case, it is desirable to have a notion of efficient universal quantum circuits in the spirit of universal quantum Turing machines.

Like resource-bounded universal Turing machines, an efficiently constructed universal circuit for a complexity class defined by resource bounds (depth, size, gate width, etc.) provides an upper bound on the resources needed to compute any circuit in that class. More precisely, the specific, efficient construction of a universal circuit for a class of circuits with a fixed input length, yields a single circuit which can be used to carry out the computation of every circuit in that class, basically a chip or processor for that class of circuits. The more efficient the construction of the universal circuit, the smaller and faster the processor

* Partially supported by the National Security Agency (NSA) and Advanced Research and Development Agency (ARDA) under Army Research Office (ARO) contract number DAAD 19-02-1-0058.

** Partially supported by NSF grant CCF-05-15269.

*** Partially supported by the NSA and ARDA under ARO contract number DAAD 19-02-1-0058.

for that class. For example, depth-universal circuits are desirable because they can simulate any circuit within a constant slow-down factor and hence are as time-efficient as possible.

Universal quantum circuits have been studied before in different contexts. Most of the research on universal quantum circuit classes deals with finding universal sets of gates which can be used to efficiently simulate any quantum computation ([NC00], [SR07]). Our goal is quite different; we want to create a circuit which, like a computer, takes as input both a program and data and runs the program on the data. Nielsen and Chuang [NC97] considered a similar problem, although they did not focus on efficiency of the universal circuit. They showed that it is not possible for a generic universal circuit to work for all circuits of a certain input length. We avoid this problem by considering families of circuits with a certain depth or size, constructed from a fixed family of gates.

In the case of quantum circuits there are particular issues relating to the requirements that computations must be clean and reversible which come into play, and to an extent complicate the classical methods. Still, much of our motivation for this work originates with classical results due to Cook, Valiant, and others [CH85, V76]. Cook and Hoover considered depth universality and described a depth-universal uniform circuit family for circuits of depth $\Omega(\log n)$. Valiant studied size universality and showed how to construct universal circuits of size $O(s \log s)$ to simulate any circuit of size s . (See Sect. 1.1).

Definition 1 (Universal Quantum Circuits). Fix $n > 0$ and let $\mathcal{C}$ be a collection of quantum circuits on n qubits. A quantum circuit U on $n+m$ qubits is universal for $\mathcal{C}$ if, for every circuit $C \in \mathcal{C}$, there is a string $x \in \{0,1\}^m$ (the encoding) such that for all strings $y \in \{0,1\}^n$ (the data),

$$U(|y\rangle \otimes |x\rangle) = C|y\rangle \otimes |x\rangle.$$

The circuit collections we are interested in are usually defined by bounding various parameters such as the size (number of gates), depth (number of layers of gates acting simultaneously on disjoint sets of qubits), or palette of allowed gates (e.g., Hadamard, $\pi/8$, CNOT).

As in the classical case, we also want our universal circuits to be *efficient* in various ways. For one, we restrict them to using the same gate family as the circuits they simulate. We may also want to restrict their size or the number m of qubits they use for the encoding. We are particularly concerned with the depth of universal circuits. Depth-universal circuits are desirable because they can simulate any circuit within a constant slow-down factor. Thus they are as time-efficient as possible.

Definition 2 (Depth-Universal Quantum Circuits). Fix a family $\mathcal{F}$ of unitary quantum gates. A family of quantum circuits $\{U_{n,d}\}_{n,d>0}$ is depth-universal over $\mathcal{F}$ if

1. $U_{n,d}$ is universal for n -qubit circuits with depth $\leq d$ using gates from $\mathcal{F}$,
2. $U_{n,d}$ only uses gates drawn from $\mathcal{F}$,

3. $U_{n,d}$ has depth $O(d)$, and
4. the number of encoding qubits of $U_{n,d}$ is polynomial in n and d .

Our first result, presented in Sect. 3, shows that depth-universal quantum circuits exist for the gate families $\mathcal{F} = \{H, T\} \cup \{F_n \mid n \geq 1\}$ and $\mathcal{F}' = \{H, T\} \cup \{F_n \mid n \geq 1\} \cup \{\wedge_n(X) \mid n \geq 1\}$, where H and T are the Hadamard and $\pi/8$ gates, respectively, and F_n and $\wedge_n(X)$ are the $(n+1)$ -qubit fanout and $(n+1)$ -qubit Toffoli gates, respectively (see Sect. 2).

In order to construct efficient universal circuit families, it appears necessary to resort to the massive parallelism that fanout gates¹ provide; note that the existing classical constructions make abundant use of fanout which is a very natural operation for classical circuits. It is an open question whether the same efficiency can be achieved without using fanout gates.

Theorem 1. *Depth-universal quantum circuits exist over $\mathcal{F}$ and over $\mathcal{F}'$. Such circuits use $O(n^2d)$ qubits and can be built log-space uniformly in n and d .*

Note that the results for the two circuit families are independent, because it is not known whether n -qubit Toffoli gates can be implemented exactly in constant depth using single-qubit gates and fanout gates, although they can be approximated this way [HS05]. It would be nice to find depth-universal circuits over families of bounded-width gates² such as $\{H, T, \text{CNOT}\}$. A simple connectivity argument shows that depth-universal circuits with bounded-width gates, if they exist, must have depth $\Omega(\log n)$ and thus can only depth-efficiently simulate circuits with depth $\Omega(\log n)$. One can therefore only hope to find depth-universal circuits for circuits of depth $\Omega(\log n)$ over bounded-width gates. Although such circuits exist in the classical case (see below), we are unable to construct them in the quantum case (see Sect. 5).

1.1 Other Relevant Work

Cook and Hoover [CH85] considered the problem of constructing general-purpose classical (Boolean) circuits using gates with fanin two. They asked whether, given n, c, d , there is a circuit U of size $c^{O(1)}$ and depth $O(d)$ that can simulate any n -input circuit of size c and depth d . Cook and Hoover constructed a depth-universal circuit for depth $\Omega(\log n)$ and polynomial size, but which takes as input a nonstandard encoding of the circuit, and they also presented a circuit with depth $O(\log n \log \log n)$ to convert the standard encoding of the circuit to the required encoding. Valiant looked at a similar problem—trying to minimize the size of the universal circuit [V76]. He considered classical circuits built from fanin 2 gates (but with unbounded fanout) and embedded the circuit in a larger universal graph. He managed to create universal graphs for different types of circuits and showed how to construct a $O(c \log c)$ -size and $O(c)$ -depth universal circuit. He also showed that his constructions have size within a constant multiplicative factor of the information theoretic lower bound.

¹ The fanout gate is a quantum analog of the classical fanout operation. See Sect. 2.

² The width of a gate is the number of qubits it acts upon.

For quantum circuits, Nielsen and Chuang (in [NC97]) considered the problem of building what they call *programmable universal gate arrays*. These generic universal circuits work on two quantum registers, a data register and a program register. They do not consider any size or depth bound on the circuits and show that it is not possible to have a generic universal circuit which works for all circuits of a certain input length. However they showed that it is possible to construct an extremely weak type of probabilistic universal circuit with size linear in the number of inputs to the simulated circuit.

Sousa and Ramos considered a similar problem of creating a universal quantum circuit to simulate any quantum gate [SR07]. They construct a basic building block which can be used to implement any single-qubit or CNOT gate on n qubits by switching certain gates on and off. They showed how to combine several of these building blocks to implement any n -qubit quantum gate.

2 Preliminaries

For the rest of the paper, we will use U to denote the universal circuit and C to denote the circuit being simulated. We assume the standard notions of quantum states, quantum circuits, and quantum gates described in [NC00], in particular, H (Hadamard), T ($\pi/8$), $S = T^2$ (phase), and CNOT (controlled NOT). We will also need some additional gates, which we now motivate.

The depth-universal circuits we construct require the ability to feed the output of a single gate to many other gates. While this operation, commonly known as fanout, is common in classical circuits, copying an arbitrary quantum state unitarily is not possible in quantum circuits due to the no-cloning theorem [NC00]. It turns out that we can construct our circuits using a classical notion of fanout operation, defined as the *fanout gate*

$$F_n : |c, t_1, \dots, t_n\rangle \mapsto |c, c \oplus t_1, \dots, c \oplus t_n\rangle$$

for any of the standard basis states $|c\rangle$ (the control) and $|t_1\rangle, \dots, |t_n\rangle$ (the targets) and extended linearly to other states³ [FFGHZ06]. F_n can be constructed in depth $\lg n$ using CNOT gates. We need to use unbounded fanout gates to achieve full depth universality. We also use the *unbounded Toffoli gate* $\wedge_n(X) : |c_1, \dots, c_n, t\rangle \mapsto |c_1, \dots, c_n, t \oplus \bigwedge_{i=1}^n c_i\rangle$. We reserve the term “Toffoli gate” to refer to the (standard) Toffoli gate $\wedge_2(X)$, which is defined on three qubits.

In addition to the fanout gate, our construction requires us to use controlled versions of the gates used in the simulated circuit. For most of the commonly used basis sets of gates (e.g., Toffoli gate, Hadamard gate, and phase gate S), the gates themselves are sufficient to construct their controlled versions (e.g., a controlled Hadamard gate can be constructed using a Toffoli gate and Hadamard and phase gates). Depth or size universality requires that the controlled versions of the gates should be constructible using the gates themselves within proper depth or size, as required.

³ This does not contradict the no-cloning theorem as only classical states are copied.

Definition 3 (Closed under controlled operation). *A set of quantum gates $G = \{G_1, \dots\}$ is said to be closed under controlled operation if for each $G_i \in G$, the controlled version of the gate $C\text{-}G_i|c\rangle|t\rangle \longrightarrow |c\rangle G_i^c|t\rangle$ can be implemented in constant depth and size using the gates in G . Here, $|c\rangle$ is a single qubit and G_i could be a single or a multi-qubit gate.*

Note that $\text{CNOT} = F_1$, and given H , T , and CNOT we can implement the Toffoli gate via a standard constant-size circuit [NC00]. We can implement the phase gate S as T^2 , and since $T^8 = I$, we can implement $S^\dagger = T^6$ and $T^\dagger = T^7$.

A *generalized Z gate*, which we will hereafter refer to simply as a *Z gate*, is an extension of the single-qubit Pauli Z gate ($|x\rangle \mapsto (-1)^x|x\rangle$) to multiple qubits:

$$|x_1, \dots, x_n\rangle \xrightarrow{Z} (-1)^{x_1 x_2 \dots x_n} |x_1, \dots, x_n\rangle.$$

A Z gate can be constructed easily (in constant depth and size) from a single unbounded Toffoli gate (and vice versa) by conjugating the target qubit of the unbounded Toffoli gate with H gates (i.e., placing H on both sides of the Toffoli gate on its target qubit).

Similarly, a *Z-fanout gate* Z_n applies the single-qubit Z gate to each of n target qubits if the control qubit is set:

$$|c, t_1, \dots, t_n\rangle \xrightarrow{Z_n} (-1)^{c \cdot (t_1 + \dots + t_n)} |c, t_1, \dots, t_n\rangle.$$

A Z_n gate can be constructed from a single F_n gate and vice versa in constant depth (although not constant size) by conjugating each target with H gates. So, in our depth-universal circuit construction, we can use either or both of these types of gates. Similarly for unbounded Toffoli versus Z gates. Z gates and Z -fanout gates are convenient to work with because they only change the phase, leaving the values of the qubits intact (they are represented by diagonal matrices in the computational basis). This allows us to use a trick due to Høyer and Špalek [HS05] and run all possible gates for a layer in parallel.

3 Depth-Universal Quantum Circuits

In this section, we prove Theorem 1, i.e., that depth-universal circuits exist for each of the gate families

$$\begin{aligned} \mathcal{F} &= \{H, T\} \cup \{F_n \mid n \geq 1\}, \\ \mathcal{F}' &= \{H, T\} \cup \{F_n \mid n \geq 1\} \cup \{\wedge_n(X) \mid n \geq 1\}. \end{aligned}$$

We first give the proof for $\mathcal{F}$ then show how to modify it for $\mathcal{F}'$.

The depth-universal circuit U we construct simulates the input circuit C layer by layer, where a layer consists of the collection of all its gates at a fixed depth. C is encoded in a slightly altered form, however. First, all the fanout gates in C are replaced with Z -fanout gates on the same qubits with H gates conjugating

the targets. At worst, this may roughly double the depth of C (adjacent H gates cancel). Each layer of the resulting circuit is then separated into three adjacent layers: the first having only the H gates of the original layer, the second only the T gates, and the third only the Z -fanout gates. U then simulates each layer of the modified C by a constant number of its own layers. We describe next how these layers are constructed.

Simulating single-qubit gates. The circuit to simulate an n -qubit layer of single-qubit gates of type G , say, consists of a layer of controlled- G gates where the control qubits are fed from the encoding and the target qubits are the data qubits. Figure 1 shows a layer of G gates, where $G \in \{H, T\}$, controlled using H , S , T , CNOT, and Toffoli gates. To simulate G gates on qubits $i_1, \dots, i_k$, say, set $c_{i_1}, \dots, c_{i_k}$ to 1 and the rest of the c -qubits to 0.

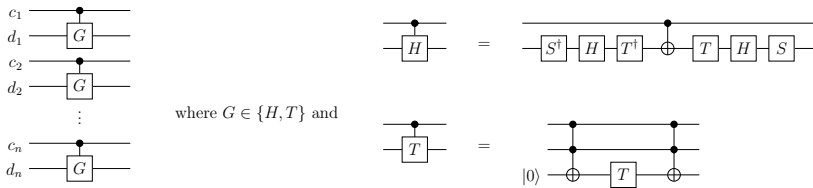


Fig. 1. Simulating a layer of single-qubit G gates with controlled G gates. The ancilla is part of the encoding and can be reused for implementing all T layers.

Simulating Z -fanout gates. The circuit to simulate a Z -fanout layer is shown in Figure 2.

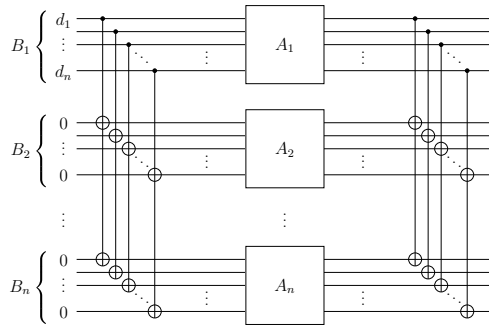


Fig. 2. Simulating a layer of Z -fanout gates

The top n qubits are the original data qubits. The rest are ancilla qubits. All the qubits are arranged in n blocks $B_1, \dots, B_n$ of n qubits per block. The qubits in block B_i are labeled $b_{i1}, \dots, b_{in}$. Each A_i subcircuit looks like Figure 3.

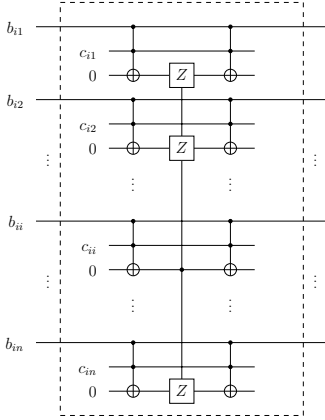


Fig. 3. Subcircuit A_i in the simulation of Z -fanout gates

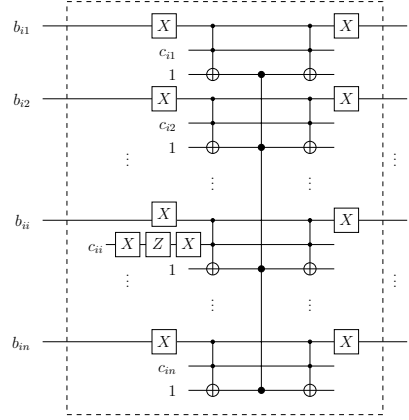


Fig. 4. Subcircuit A_i for a layer of Z gates

The qubits $c_{i1}, \dots, c_{in}$ are encoding qubits. The large gate between the two columns of Toffoli gates is a Z -fanout gate with its control on the i th ancilla (corresponding to b_{ii} and c_{ii}) and targets on all the other ancillæ.

Here is the state evolution from $|\mathbf{d}\rangle = |d_1 \dots d_n\rangle$, suppressing the c_{ij} qubits and ancillæ internal to the A_i subcircuits in the ket labels. Note that after the first layer of fanouts, each qubit b_{ij} carries the value d_j .

$$\begin{aligned}
 |\mathbf{d}, \mathbf{0}, \dots, \mathbf{0}\rangle &\mapsto |\mathbf{d}, \mathbf{d}, \dots, \mathbf{d}\rangle \\
 &\mapsto (-1)^{\sum_i d_i c_{ii} (\sum_{j \neq i} d_j c_{ij})} |\mathbf{d}, \mathbf{d}, \dots, \mathbf{d}\rangle \\
 &\mapsto (-1)^{\sum_i d_i c_{ii} (\sum_{j \neq i} d_j c_{ij})} |\mathbf{d}, \mathbf{0}, \dots, \mathbf{0}\rangle.
 \end{aligned}$$

To simulate some Z -fanout gate G of C whose control is on the i th qubit, say, we do this in block B_i by setting c_{ii} to 1 and c_{ij} to 1 for every j where the j th qubit is a target of G . All the other c -qubits in B_i are set to 0. We can do this in separate blocks for multiple Z -fanout gates on the same layer, because no two gates can share the same control qubit. Any c -qubits in unused blocks are set to 0.

Simulating unbounded Toffoli gates. We modify the construction above to accommodate unbounded Toffoli gates (family $\mathcal{F}'$), or equivalently Z gates, by breaking each layer of C into four adjacent layers, the first three being as before, and the fourth containing only Z gates. The top-level circuit to simulate a layer of Z gates is as before (Figure 2) and each A_i subcircuit looks like Figure 4, where the central gate is a Z gate connecting the ancillæ. To simulate a Z gate G of C whose first qubit is i , say, we do this in block B_i by setting c_{ii} to 1 and setting c_{ij} to 1 for every j where the j th qubit is part of G . All the other c -qubits in B_i are set to 0. As before, we do this in separate blocks for multiple gates on the same layer, since no two gates can share the same first qubit. Any c -qubits in unused blocks are set to 0, and it is easy to check that this makes the block have no net effect.

4 Size-Universal Quantum Circuits

Similar to a depth-universal circuit, a *size-universal circuit* is a universal circuit with the same order of the number of gates as the circuit it is simulating. Formally,

Definition 4. A family $\{U_{n,c}\}$ of universal circuits for n -qubit circuits of size $\leq c$ is size-universal if $SIZE(U_{n,c}) = O(c)$.

Via a simple counting argument, it is not possible to obtain a completely size-universal circuit for fanin-2 circuits. The number of possible circuits with c fanin-2 gates is $\Omega((n-1)^{c+1})$. Since all the encoding bits have to be connected to some of the fanin-2 gates in the universal circuit, it must have $\Omega(c \log n)$ gates.

We use Valiant's idea of universal graphs [V76] to construct a universal family of fanin-2 circuits that are very close to the aforementioned lower bound. As before, we would like to simulate C by using the same set of gates used in C . Our construction works for any circuit using unbounded Toffoli gates and any set of single-qubit and 2-qubit gates closed under the controlled operation.

The graph of any circuit of size n can be represented as a directed acyclic graph with vertices $\{1, \dots, n\}$ such that there is no edge from j to i for $i < j$ and each vertex has fanin and fanout 2. Let $\Gamma_2(n)$ be the set of all such graphs.

Definition 5 (Edge-universal graph [V76]). A graph G' is edge-universal for $\Gamma_2(n)$ if it has distinct vertices (poles) $p_1, \dots, p_n$ such that any graph $G \in \Gamma_2(n)$ can be embedded into G' where each vertex $i \in G$ is mapped to distinct vertex $\rho(i) = p_i \in G'$ and distinct edges $(i, j) \in E$ are mapped to edge-disjoint paths $\rho(i) \rightsquigarrow \rho(j) \in E'$.

Theorem 2 (Universal graph[V76]). There is a constant k such that for all n there exists an acyclic graph G' that is edge-universal for $\Gamma_2(n)$, and G' has $kn \lg n$ vertices, each vertex having fanin and fanout 2.

It is fairly easy to construct a universal circuit using the universal graph. Consider any edge-universal graph G' for $\Gamma_2(n+c)$. Then G' has $c' = k(n+c) \log(n+c)$ vertices for some k . These c' vertices include poles $p_1, \dots, p_{n+c}$ and non-pole vertices. Create a corresponding quantum circuit C' with c' gates (including the inputs). For each of the vertices $p_1, \dots, p_n$ of G' , remove their incoming edges and replace the vertices by the input as shown in Figure 5. Replace each of the vertices $p_{n+1}, \dots, p_{n+c}$ with a subcircuit that applies any of the single- or 2-qubit gates on the inputs, where the gate to apply is controlled by the encoding (Figure 7 shows an example). For a non-pole vertex, replace it with a subcircuit that swaps the incoming and outgoing wires (i.e., first input is connected to second output and second input is connected to first output) or directly connects them (i.e., first input is connected to first output and similarly for the second input) depending on the encoding (see Figure 6). The edge disjointness property guarantees that wires in the embedded circuit are mapped to paths in C' which can share a vertex but cannot share any edge.

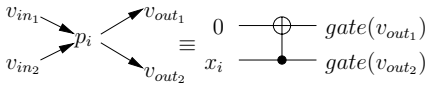


Fig. 5. The gate for a pole vertex p_i is mapped to input x_i

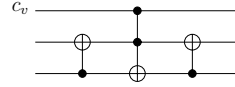


Fig. 6. The gates at a non-pole vertex v . The encoding bit c_v specifies if first output qubit should be mapped to first input or second input qubit and similarly for second output qubit.

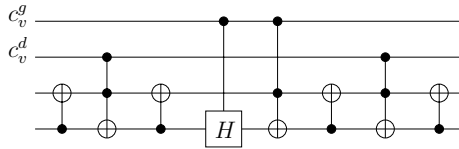


Fig. 7. Example of the gates at a pole vertex v simulating a circuit with CNOT and H gates. The encoding bit c_v^g specify the type of gate at v , and the c_v^d specify which qubit the gate acts on (for H gate) or which is the control qubit (for CNOT gate).

To simulate any fanin-2 circuit C with c gates acting on n qubits, construct the edge-universal graph G' for $\Gamma_2(n + c)$. Embed the graph of C into G' such that the input nodes of C are mapped to the poles $p_1, \dots, p_n$ in G' . Now for each gate of the circuit, set a bit in the encoding to denote the type of the gate at that pole to which it was mapped. For the non-pole vertices, set a bit in the encoding to specify whether the two input values should be swapped or mapped directly to the two output values.

Theorem 3. *There is a constant k and a family of universal circuits $U_{n,c}$ that can simulate every circuit with c gates acting on n qubits such that $\text{SIZE}(U_{n,c}) = O((n + c) \log(n + c))$.*

For unbounded fanin circuits, if we can decompose the unbounded fanin gates into linearly many bounded fanin gates, then a similar idea can be used.

Corollary 1. *There is a family of universal circuits $U_{n,c}$ that can simulate quantum circuits of size c on n qubits and consisting of Hadamard, $\pi/8$, and unbounded Toffoli gates such that $\text{SIZE}(U_{n,c}) = O(nc \log(nc))$.*

5 Conclusions

We have been mostly concerned with the actual simulation of a quantum circuit C by the universal circuit U . However, it is possible to hide some complexity of the simulation in U 's description of C itself. Usually, the description of a classical circuit is the underlying graph of the circuit and specifies the gates at each vertex. But we use a grid description of quantum circuits which is more

natural and especially suitable for simulation; in the description the rows of the grid correspond to the qubits, and the columns correspond to the different layers of the circuit. A graph-based description can be easily converted to this grid-based description in polynomial time.

The techniques of Sect. 3 can be easily adapted to build depth-universal circuits for a variety of classical (Boolean) circuit classes with unbounded gates, e.g., AC, ACC, and TC circuits. The key reason is that these big gates are all “self-similar” in the sense that fixing some of the inputs can yield a smaller gate of the same type.

A number of natural, interesting open problems remain.

Fanout gates are used in our construction of a depth-universal circuit family. Is the fanout gate necessary in our construction? We believe it is. In fact, we do not know how to simulate depth- d circuits over $\{H, T, \text{CNOT}\}$ universally in depth $O(d)$ without using fanout gates, even assuming that the circuits being simulated have depth $\Omega(\log n)$. The shallowest universal circuits with bounded-width gates we know of have a $\lg n$ blow-up factor in the depth, just by replacing the fanout gates with log-depth circuits of CNOT gates.

Our results apply to circuits with very specific gate sets. How much can these gate sets be generalized? Are similar results possible for any countable set of gates containing Hadamard, unbounded Toffoli, and fanout gates?

We showed how to construct a universal circuit with a logarithmic blow-up in size. The construction is within a constant factor of the minimum possible size for polynomial-size, bounded-fanin circuits. However for constant-size circuits, we believe the lower bound can be tightened to match the proven upper bound.

Acknowledgments

We thank Michele Mosca and Debbie Leung for insightful discussions. The second author is grateful to Richard Cleve and IQC (Waterloo) and to Harry Buhrman and CWI (Amsterdam) for their hospitality.

References

- [D85] Deutsch, D.: Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. In: Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, vol. 400, pp. 97–117 (1985)
- [BGH07] Bera, D., Green, F., Homer, S.: Small depth quantum circuits. SIGACT News 38(2), 35–50 (2007)
- [CH85] Cook, S.A., Hoover, H.J.: A depth-universal circuit. SIAM Journal on Computing 14(4), 833–839 (1985)
- [FFGHZ06] Fang, M., Fenner, S., Green, F., Homer, S., Zhang, Y.: Quantum lower bounds for fanout. Quantum Information and Computation 6(1), 46–57 (2006)
- [HS05] Høyer, P., Špalek, R.: Quantum circuits with unbounded fan-out. Theory of Computing 1, 81–103 (2005)

- [NC97] Nielsen, M.A., Chuang, I.L.: Programmable quantum gate arrays. *Phys. Rev. Lett.* 79(2), 321–324 (1997)
- [NC00] Nielsen, M.A., Chuang, I.L.: *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge (2000)
- [SR07] Sousa, P.B.M., Ramos, R.V.: Universal quantum circuit for n -qubit quantum gate: A programmable quantum gate. *Quantum Information and Computation* 7(3), 228–242 (2007)
- [V76] Valiant, L.G.: Universal circuits (preliminary report). In: *Proceedings of the 8th ACM Symposium on the Theory of Computing*, pp. 196–203 (1976)
- [Y93] Yao, A.C.C.: Quantum circuit complexity. In: *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pp. 352–361 (1993)

An Improved Time-Space Lower Bound for Tautologies

Scott Diehl^{1,*}, Dieter van Melkebeek^{2,**}, and Ryan Williams^{3,***}

¹ Computer Science Department, Siena College, Loudonville, NY 12211
sfdiehl@siena.edu

² Computer Sciences Department, University of Wisconsin, Madison, WI 53706
dieter@cs.wisc.edu

³ School of Mathematics, Institute for Advanced Study, Princeton, NJ 08540
ryanw@ias.edu

Abstract. We show that for all reals c and d such that $c^2d < 4$ there exists a real $e > 0$ such that tautologies of length n cannot be decided by both a nondeterministic algorithm that runs in time n^c , and a nondeterministic algorithm that runs in time n^d and space n^e . In particular, for all $d < \sqrt[3]{4}$ there exists an $e > 0$ such that tautologies cannot be decided by a nondeterministic algorithm that runs in time n^d and space n^e .

1 Introduction

Proof complexity studies the NP versus coNP problem — whether tautologies can be recognized efficiently by nondeterministic machines. Typical results in proof complexity deal with specific types of nondeterministic machines that implement well-known proof systems, such as resolution. They establish strong (superpolynomial or even exponential) lower bounds for the size of any proof of certain families of tautologies within that system, and thus for the running time of the corresponding nondeterministic machine deciding tautologies.

Another, more generic, approach to the NP versus coNP problem follows along the lines of the recent time-space lower bounds for satisfiability on deterministic machines [4]. Similar arguments yield lower bounds for satisfiability on conondeterministic machines, or equivalently, for tautologies on nondeterministic machines. Those results show that no nondeterministic algorithm can decide tautologies in time n^d and space n^e for interesting combinations of d and e . The lower bounds obtained are very robust with respect to the model of computation and apply to any proof system. However, the arguments only work in the polynomial time range (constant d) and sublinear space range ($e < 1$). For example, Fortnow [1] proved that we must have $d > 1$ whenever $e < 1$, and Fortnow and

* Research partially supported by NSF award CCR-0728809 and a Cisco Systems Distinguished Graduate Research Fellowship while at the University of Wisconsin.

** Research partially supported by NSF award CCR-0728809 and partially performed while visiting the Weizmann Institute of Science.

*** Research partially supported by NSF award CCF-0832797.

Van Melkebeek [3] (see also [2]) showed a time lower bound of n^d for any $d < \sqrt{2}$ in the case of subpolynomial space bounds ($e = o(1)$).

In this paper we build on these generic techniques and boost the exponent in the time lower bound for subpolynomial-space nondeterministic algorithms recognizing tautologies from $\sqrt{2} \approx 1.414$ to $\sqrt[3]{4} \approx 1.587$.

Theorem 1. *For every real $d < \sqrt[3]{4}$ there exists a positive real e such that tautologies cannot be decided by nondeterministic algorithms running in time n^d and space n^e .*

The earlier result of Fortnow and Van Melkebeek [3] can be refined to rule out either nondeterministic algorithms solving tautologies in time n^c (regardless of space) or nondeterministic algorithms solving tautologies in simultaneous time n^d and space n^e for certain combinations of c , d , and e . More precisely, for every c and d such that $(c^2 - 1)d < c$, there is an $e > 0$ satisfying the lower bound. For example, tautologies cannot have both a nondeterministic algorithm using $n^{1+o(1)}$ time and a nondeterministic algorithm using logarithmic space [1]. Correspondingly, our argument yields the following refinement.

Theorem 2. *For all reals c and d such that $c^2d < 4$, there exists a positive real e such that tautologies of length n cannot be solved by both*

- (i) *a nondeterministic algorithm that runs in time n^c and*
- (ii) *a nondeterministic algorithm that runs in time n^d and space n^e .*

The interesting range of parameters in Theorem 2 is $d \geq c \geq 1$, since an algorithm of type (ii) is a special case of an algorithm of type (i) for $d \leq c$, and a sublinear-time algorithm can be ruled out unconditionally by simple diagonalization. The condition due to this paper, $c^2d < 4$, is less restrictive for values of d that are close to c . In particular, for $c = d$, our condition requires $d < \sqrt[3]{4} \approx 1.587$, whereas that of [3] requires $d < \sqrt{2} \approx 1.414$; this setting is the improvement stated in Theorem 1.

Our main technical contribution is another level of sophistication in the indirect diagonalization paradigm, corresponding to the transition from linear to nonlinear dynamics. We start from the hypothesis that tautologies have machines of types (i) and (ii), and aim to derive a contradiction. Fortnow and Van Melkebeek [3] use (ii) to obtain a nondeterministic time-space efficient simulation of conondeterministic computations. Next, they speed up the space-bounded nondeterministic computation à la Savitch [5] by introducing alternations, and subsequently eliminate those alternations efficiently using (i). When $(c^2 - 1)d < c$, the net effect is a speedup of generic conondeterministic computations on nondeterministic machines, implying the sought-after contradiction.

The above argument exploits (ii) in a rather limited way, namely only in the very first step. One could use (ii) instead of (i) to eliminate alternations. Since $d \geq c$ this costs at least as much time as using (i), but the space bound induced by (ii) allows us to run another layer of alternation-based speedups and alternation eliminations. Due to the additional layer, the recurrence relation for the net speedup becomes of degree two (rather than one as before) and has

nonconstant coefficients, but we can still handle it analytically. We point out that this is the first application of nonlinear dynamics in analyzing time-space lower bounds for satisfiability and related problems.

2 Preliminaries

2.1 Notation

For functions t and s we denote by $\text{NTIME}(t)$ the class of languages recognized by nondeterministic machines that run in time $O(t)$, and by $\text{NTISP}(t, s)$ those recognized by nondeterministic machines that run in simultaneous time $O(t)$ and space $O(s)$. We use the prefix “co” to represent the complementary classes. We often use the same notation to refer to classes of machines rather than languages.

Our results are robust with respect to the choice of machine model underlying our complexity classes; for concreteness, we use the random-access machine model as described in [4]. Note that all instances of t and s in this paper are polynomials in n , so they are easily constructible.

Recall that a space-bounded nondeterministic machine does not have two-way access to its guess bits unless it explicitly writes them down on its worktape at the expense of space. It is often important for us to take a finer-grained view of such computations to separate out the resources required to write down a nondeterministic guess string from those required to verify that the guess is correct. To this end, we adopt the following notation.

Definition 1. *Given a complexity class $\mathcal{C}$ and a function f , we define the class $\exists^f \mathcal{C}$ to be the set of languages that can be described as*

$$\{x \mid \exists y \in \{0, 1\}^{O(f(|x|))} P(x, y)\},$$

where P is a predicate accepting a language in the class $\mathcal{C}$ when its complexity is measured in terms of $|x|$ (not $|x| + |y|$). We analogously define $\forall^f \mathcal{C}$.

2.2 Tautologies versus Conondeterministic Linear Time

All known time-space lower bounds for satisfiability or tautologies hinge on the tight connection between the tautologies problem and the class of languages recognized by conondeterministic linear-time machines, $\text{coNTIME}(n)$. Strong versions of the Cook-Levin Theorem have been formulated, showing that the tautologies problem captures the simultaneous time *and* space complexity of conondeterministic linear time on nondeterministic machines, up to polylogarithmic factors. As a consequence, time-space lower bounds for $\text{coNTIME}(n)$ on nondeterministic machines transfer to tautologies with little loss in parameters. In particular we use the following result; see [4] for an elementary proof.

Lemma 1. *For positive reals d and e , if*

$$\text{coNTIME}(n) \not\subseteq \text{NTISP}(n^d, n^e),$$

then for any reals $d' < d$ and $e' < e$,

$$\text{Tautologies} \notin \text{NTISP}(n^{d'}, n^{e'}).$$

Since a lower bound for $\text{coNTIME}(n)$ yields essentially the same lower bound for tautologies, we shift our focus to proving lower bounds for the former.

2.3 Indirect Diagonalization

Our proofs follow the paradigm of indirect diagonalization. The paradigm works by contradiction. In the case of Theorem 2 we assume that

$$\text{coNTIME}(n) \subseteq \text{NTIME}(n^c) \cap \text{NTISP}(n^d, n^e). \quad (1)$$

This unlikely assumption is used to derive more and more unlikely inclusions of complexity classes, until some inclusion contradicts a known diagonalization result. The main two tools we use to derive inclusions go in opposite directions:

- (a) Speed up nondeterministic space-bounded computations by adding alternations, and
- (b) Eliminate these alternations via assumption (1), at a moderate increase in running time.

To envision the utility of these items, notice that (1) allows the simulation of a conondeterministic machine by a space-bounded nondeterministic machine. Item (a) allows us to simulate the latter machine by an alternating machine that runs in less time. Using item (b), the alternations can be eliminated from this simulation, increasing the running time modestly. In this way, we end up back at a nondeterministic computation, so that overall we have derived a simulation of a conondeterministic machine by a nondeterministic one. The complexity class inclusion that this simulation yields is a complementation of the form

$$\text{coNTIME}(t) \subseteq \text{NTIME}(f(t)), \quad (2)$$

where we seek to make f as small as possible by carefully compounding applications of (a) and (b). In fact, we know how to rule out inclusions of the type (2) for small functions f , say $f(t) = t^{1-\epsilon}$, by a folklore diagonalization argument. This supplies us with a means for deriving a contradiction.

Lemma 2. *Let a and b be positive reals such that $a < b$, then*

$$\text{coNTIME}(n^b) \not\subseteq \text{NTIME}(n^a).$$

Let us discuss how to achieve items (a) and (b). Item (a) is filled in by the divide-and-conquer strategy that underlies Savitch's Theorem [5]. Briefly, the idea is to divide the rows in a computation tableau of a space-bounded nondeterministic machine M into b time blocks. Observe that M accepts x in time t if and only if there are $b-1$ configurations $C_1, C_2, \dots, C_{b-1}$ at the boundaries of these blocks such that for every block i , $1 \leq i \leq b$, the configuration at the beginning of that

block, C_{i-1} , can reach the configuration at the end of that block, C_i , in t/b steps, where C_0 is the initial configuration and C_b is the accepting configuration. This condition is implemented on an alternating machine to realize a speedup of M as follows. First existentially guess $b - 1$ configurations of M , universally guess a block number i , and decide if C_{i-1} reaches C_i via a simulation of M for t/b steps. Thus, we can derive that

$$\text{NTISP}(t, s) \subseteq \exists^{bs} \forall^{\log b} \text{NTISP}(t/b, s). \quad (3)$$

The above simulation runs in overall time $O(bs + t/b)$. Choosing $b = O(\sqrt{t/s})$ minimizes this running time, to $O(\sqrt{ts})$. However, this minimization produces suboptimal results in our arguments. Instead, we apply (3) for an unspecified b and choose the optimal value after all of our derivations.

Let us point out one important fact about the simulation underlying (3). The final phase of this simulation, that of simulating M for t/b steps, does not need access to all of the configurations guessed during the initial existential phase — it only reads the description of two configurations, C_{i-1} and C_i , in addition to the original input x . Thus, the input size of the final stage is $O(n + s)$, as opposed to $O(n + bs)$ as the complexity-class inclusion of (3) suggests in general. This fact has a subtle but key impact in our lower bound proof.

We now turn to item (b), that of eliminating the alternations introduced by (3). In general, eliminating alternations comes at an exponential cost. However, in our case we are armed with assumption (1). The assumption $\text{coNTIME}(n) \subseteq \text{NTIME}(n^c)$ allows us to eliminate an alternation at the cost of raising the running time to the power of c . Alternately, assuming $\text{coNTIME}(n) \subseteq \text{NTISP}(n^d, n^e)$ allows us to eliminate an alternation at the cost of raising the running time to the power of d while at the same time maintaining the space restriction of $O(n^e)$ on the final stage. We use both of these ideas in our argument.

3 Proof of the Lower Bound

We begin with a brief discussion of the strategy used to prove the condition $(c^2 - 1)d < c$ of [3]. The relevant technical lemma from [3] can be thought of as trading space for time within NP under the indirect diagonalization assumption (1). More precisely, it tries to establish

$$\text{NTISP}(t, s) \subseteq \text{NTIME}(g(t, s)) \quad (4)$$

for the smallest possible functions g , with the hope that $g(t, s) \ll t$. In particular, for subpolynomial space bounds ($s = t^{o(1)}$) and sufficiently large polynomial t , [3] achieves $g = t^{c-1/c+o(1)}$,

$$\text{NTISP}(t, t^{o(1)}) \subseteq \text{NTIME}(t^{c-1/c+o(1)}), \quad (5)$$

which is smaller than t when $c < \phi \approx 1.618$.

As an example of the utility of inclusion (5), let us sketch the $n^{\sqrt{2}-o(1)}$ lower bound of [3] for subpolynomial-space nondeterministic algorithms solving tau-tologies. We assume, by way of contradiction, that

$$\text{coNTIME}(n) \subseteq \text{NTISP}(n^c, n^{o(1)}). \quad (6)$$

Then, for sufficiently large polynomials t , we have that:

$$\begin{aligned} \text{coNTIME}(t) &\subseteq \text{NTISP}(t^c, t^{o(1)}) && \text{[by assumption (6)]} \\ &\subseteq \text{NTIME}(t^{c^2-1+o(1)}) && \text{[by trading space for time using (5)].} \end{aligned}$$

This contradicts Lemma 2 when $c < \sqrt{2}$, yielding the desired lower bound.

The space-for-time inclusion (5) is shown by an inductive argument that derives statements of the type (4) for a sequence of smaller and smaller running times $\{g_\ell\}$. The idea can be summarized as follows. We start with a space-bounded nondeterministic machine and apply the speedup (3), yielding

$$\text{NTISP}(t, s) \subseteq \underbrace{\exists^{bs} \forall^{\log b} \underbrace{\text{NTISP}(t/b, s)}_{(7a)}}_{(7b)}. \quad (7)$$

The inductive hypothesis is then applied to trade the space bound of the final stage (7a) of this Σ_3 -simulation for time:

$$\text{NTISP}(t, s) \subseteq \exists^{bs} \forall^{\log b} \text{NTIME}(g_{\ell-1}(t/b, s)).$$

Finally, we use assumption (6) to eliminate the two alternations in this simulation, ending up with another statement of the form

$$\text{NTISP}(t, s) \subseteq \text{NTIME}(g_\ell(t, s)).$$

Notice that the above argument does not rely on the space bound in (6); the weaker assumption that $\text{coNTIME}(n) \subseteq \text{NTIME}(n^c)$ is enough to eliminate the alternations introduced by the speedup. Our new argument does exploit the fact that when we transform (7a) using the assumption (6), we eliminate an alternation *and* re-introduce a space-bound. This allows us to apply the inductive hypothesis for a *second time* and trade the space bound for a speedup in time once more. This way, we hope to eliminate the alternation in (7b) more efficiently than before, yielding a smaller g_ℓ after completing the argument.

Some steps of our argument exploit the space bound while others do not. We allow for different parameters in those two types of steps; we assume

$$\text{coNTIME}(n) \subseteq \text{NTISP}(n^c) \cap \text{NTISP}(n^d, n^{o(1)}),$$

where $d \geq c \geq 1$. The success of our approach to eliminate the alternation in (7b) now depends on how large d is compared with c . If d is close to c , then the increased cost of complementing via the space-bounded assumption is counteracted by the benefit of trading this space bound for time.

Two key ingredients that allow the above idea to yield a quantitative improvement for certain values of c and d are (i) that the conondeterministic guess at the beginning of stage (7b) is only over $\log b$ bits and (ii) the fact mentioned in Section 2 that (7a) has input size $O(n + s)$. Because of (i), the running time of (7b) is dominated by that of (7a), allowing us to reduce the cost of simulating (7b) without an alternation by reducing the cost of complementing (7a) into coNP. Item (ii) is important for the latter task because the effective input size for the computation (7a) is much smaller than the $O(n + bs)$ bits taken by (7b); in particular, it does not increase with b . This allows the use of larger block numbers b to achieve greater speedups while maintaining that the final stage runs in time at least linear in its input. The latter behavior is crucial in allowing alternation removal at the expected cost — raising the running time to the power of c or d — because we can pad the indirect diagonalization assumption (1) up (to superlinear time) but not down (to sublinear time).

Now that we have sketched the intuition and key ingredients, we proceed with the actual argument. The following lemma formalizes the inductive process of speeding up nondeterministic space-bounded computations on space-unbounded nondeterministic machines.

Lemma 3. *If*

$$\text{coNTIME}(n) \subseteq \text{NTIME}(n^c) \cap \text{NTISP}(n^d, n^e)$$

for some reals c, d , and e then for every nonnegative integer ℓ , time function t , and space function $s \leq t$,

$$\text{NTISP}(t, s) \subseteq \text{NTIME}((ts^\ell)^{\gamma_\ell} + (n + s)^{a_\ell}),$$

where $\gamma_0 = 1$, $a_0 = 1$, and γ_ℓ and a_ℓ are defined recursively for $\ell > 0$ as follows: Let

$$\mu_\ell = \max(\gamma_\ell(d + e\ell), ea_\ell), \tag{8}$$

then

$$\gamma_{\ell+1} = c\gamma_\ell\mu_\ell/(1 + \gamma_\ell\mu_\ell), \tag{9}$$

and

$$a_{\ell+1} = ca_\ell \cdot \max(1, \mu_\ell). \tag{10}$$

Proof. The proof is by induction on ℓ . The base case $\ell = 0$ is trivial. To argue the inductive step, $\ell \rightarrow \ell + 1$, we consider a nondeterministic machine M running in time t and space s and construct a faster simulation at the cost of sacrificing the space bound. We begin by simulating M in the third level of the polynomial-time hierarchy via the speedup (3) using $b > 0$ blocks (to be determined later); this simulation is in

$$\exists^{bs\sqrt{\log t}} \underbrace{\text{NTISP}(t/b, s)}_{(11a)}. \tag{11}$$

We focus on simulating the computation of (11a). Recall the input to (11a) consists of the original input x of M as well as two configuration descriptions of

size $O(s)$, for a total input size of $O(n + s)$. The inductive hypothesis allows the simulation of (11a) in

$$\text{NTIME} \left(\left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell} + (n + s)^{a_\ell} \right) \right). \quad (12)$$

In turn, this simulation can be complemented while simultaneously introducing a space bound via the assumption of the lemma; namely, (12) is in

$$\text{coNTISP} \left(\left(\left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell} + (n + s)^{a_\ell} \right)^d, \left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell} + (n + s)^{a_\ell} \right)^e \right) \right),$$

where here the $(n + s)^{a_\ell}$ term subsumes the $O(n + s)$ term from the input size because $a_\ell \geq 1$. The space bound allows for a simulation via the inductive hypothesis once more, yielding a simulation of (11a) in

$$\begin{aligned} \text{coNTIME} \left(\left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell} + (n + s)^{a_\ell} \right)^{\gamma_\ell(d+e\ell)} + \left(n + s + \left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell} + (n + s)^{a_\ell} \right)^e \right)^{a_\ell} \right) \\ \subseteq \text{coNTIME} \left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell \mu_\ell} + (n + s)^{a_\ell \mu_\ell} + (n + s)^{a_\ell} \right). \end{aligned} \quad (13)$$

Replacing (11a) in (11) by (13) eliminates an alternation, lowering the simulation of M to the second level of the polynomial hierarchy:

$$\underbrace{\exists^{bs} \forall^{\log t} \text{coNTIME} \left(\left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell \mu_\ell} + (n + s)^{a_\ell \mu_\ell} + (n + s)^{a_\ell} \right) \right)}_{(14a)} \quad (14)$$

We now complement the conondeterministic computation of (14a) via the assumption that $\text{NTIME}(n) \subseteq \text{coNTIME}(n^c)$, eliminating one more alternation. Since (14a) takes input of size $O(n + bs)$, this places (14) in

$$\begin{aligned} \exists^{bs} \text{NTIME} \left(\left(\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell \mu_\ell} + (n + s)^{a_\ell \mu_\ell} + (n + s)^{a_\ell} + (bs + n) \right)^c \right) \\ \subseteq \text{NTIME} \left(\left(\underbrace{\left(\frac{t}{b} s^\ell \right)^{\gamma_\ell \mu_\ell}}_{(15a)} + (n + s)^{a_\ell \mu_\ell} + (n + s)^{a_\ell} + \underbrace{bs}_{(15b)} \right)^c \right), \end{aligned} \quad (15)$$

where the inclusion holds by collapsing the adjacent existential phases (and the time required to guess the $O(bs)$ configuration bits is accounted for by the observation that $c \geq 1$).

We have now given a simulation of $\text{NTISP}(t, s)$ in $\text{NTIME}(\cdot)$; all that remains is to choose the parameter b . Notice that the running time of (15) has one term, (15b), that increases with b and one term, (15a), that decreases with b . The running time is minimized up to a constant factor by choosing b to equate the two terms, resulting in

$$b^* = \left(\frac{(ts^\ell)^{\gamma_\ell \mu_\ell}}{s} \right)^{1/(1+\gamma_\ell \mu_\ell)}.$$

When this value is at least 1, the running time of the simulation (15) is

$$O\left((ts^{\ell+1})^{c\gamma_\ell\mu_\ell/(1+\gamma_\ell\mu_\ell)} + (n+s)^{ca_\ell\mu_\ell} + (n+s)^{ca_\ell}\right),$$

resulting in the recurrences (9) and (10). If $b^* < 1$, then $b = 1$ is the best we can do; the desired bound still holds since in this case $(15a) + (15b) = O(s)$, which is dominated by $(n+s)^{a_{\ell+1}}$. $\square$

Applying Lemma 3, we deduce that for large enough polynomial τ ,

$$\text{coNTIME}(\tau) \subseteq \text{NTISP}(\tau^d, \tau^e) \subseteq \text{NTIME}(\tau^{(d+e\ell)\gamma_\ell} + \tau^{ea_\ell}) = \text{NTIME}(\tau^{\mu_\ell}), \quad (16)$$

which is a contradiction with Lemma 2 when $\mu_\ell < 1$. We now determine values of c , d , and e that imply this contradiction, focusing on small values of e .

Theorem 3. *For all reals c and d such that $c^2d < 4$ there exists a positive real e such that*

$$\text{coNTIME}(n) \not\subseteq \text{NTIME}(n^c) \cap \text{NTISP}(n^d, n^e).$$

Proof. The case where either $c < 1$ or $d < 1$ is ruled out by Lemma 2. For $c \geq 1$ and $d \geq 1$, assume (by way of contradiction) that

$$\text{coNTIME}(n) \subseteq \text{NTIME}(n^c) \cap \text{NTISP}(n^d, n^e)$$

for a value of e to be determined later. As noted above, the theorem's assumption in conjunction with Lemma 3 yields the complementation (16) for any integer $\ell \geq 0$ and sufficiently large polynomial bound τ .

Our goal is now to characterize the behavior of μ_ℓ in terms of c , d , and e . This task is facilitated by focusing on values of e that are small enough to smooth out the complex behavior of μ_ℓ caused by (i) the appearance of the nonconstant term $e\ell$ in the recurrence and (ii) its definition via the maximum of two functions.

We first handle item (i) by introducing a related, nicer sequence by substituting a real β (to be determined) as an upper bound for $e\ell$. Let

$$\mu'_\ell = \max(\gamma'_\ell(d + \beta), ea'_\ell), \quad (17)$$

where $\gamma'_0 = 1$, $a'_0 = 1$ and

$$\begin{aligned} \gamma'_{\ell+1} &= c\gamma'_\ell\mu'_\ell/(1 + \gamma'_\ell\mu'_\ell), \text{ and} \\ a'_{\ell+1} &= ca'_\ell \cdot \max(1, \mu'_\ell). \end{aligned}$$

As long as β behaves as intended, i.e., $e\ell \leq \beta$, we can show by induction that $\gamma_\ell \leq \gamma'_\ell$, $a_\ell \leq a'_\ell$, and $\mu_\ell \leq \mu'_\ell$. Therefore, μ'_ℓ upper bounds μ_ℓ up to a value of ℓ that depends on e , and this ℓ -value becomes large when e is very small. This allows us to use μ'_ℓ as a proxy for μ_ℓ in our analysis.

To smooth out the behavior caused by issue (ii), we point out that the first term in the definition (17) of μ'_ℓ is larger than the second when e is very small. Provided that this is the case, μ'_ℓ equals the sequence ν_ℓ defined as follows:

$$\begin{aligned} \nu_0 &= d + \beta \\ \nu_{\ell+1} &= \nu_\ell^2 c(d + \beta) / ((d + \beta) + \nu_\ell^2). \end{aligned}$$

This delivers a simpler sequence to analyze. Notice that because the underlying transformation

$$\eta \rightarrow \eta^2 c(d + \beta) / ((d + \beta) + \eta^2)$$

is increasing over the positive reals, the sequence ν_ℓ is monotone in this range. It is decreasing if and only if $\nu_1 < \nu_0$, which is equivalent to $(c - 1)(d + \beta) < 1$. Furthermore, when $c^2(d + \beta) < 4$, the transformation has a unique real fixed point at 0. Since the underlying transformation is also bounded and starts positively, the sequence ν_ℓ must decrease monotonically to 0 in this case.

Therefore, when $c^2 d < 4$ we can choose a positive β such that ν_ℓ becomes as small as we want for large ℓ . Provided that β , e , and ℓ satisfy the assumptions required to smooth out items (i) and (ii), this also gives us that μ_ℓ is small. More formally, let ℓ^* be the first value of ℓ such that $\nu_\ell < 1$. Item (i) requires that

$$e\ell^* \leq \beta. \quad (18)$$

Item (ii) requires that the first term of μ'_ℓ in (17) dominates the second up to this point, namely,

$$\gamma'_\ell(d + \beta) \geq ea'_\ell \text{ for all } \ell \leq \ell^*. \quad (19)$$

When all of these conditions are satisfied, we have that $\mu_{\ell^*} \leq \mu'_{\ell^*} = \nu_{\ell^*} < 1$, and the running time of the NTIME computation in (16) for $\ell = \ell^*$ is $O(\tau^{\mu_{\ell^*}}) = O(\tau^{\nu_{\ell^*}}) = O(\tau^{\mu_{\ell^*}})$.

Therefore, by choosing a small enough positive e to satisfy the finite number of constraints in (18) and (19), we arrive at our goal of proving that $\mu_\ell < 1$ in (16). This is a contradiction, which proves the desired lower bound. $\square$

We remark that our above analysis is tight, in the sense the above proof does not work for $c^2 d \geq 4$. The details will appear in the full version of the paper.

References

1. Fortnow, L.: Time-space tradeoffs for satisfiability. *Journal of Computer and System Sciences* 60, 337–353 (2000)
2. Fortnow, L., Lipton, R., van Melkebeek, D., Viglas, A.: Time-space lower bounds for satisfiability. *Journal of the ACM* 52, 835–865 (2005)
3. Fortnow, L., van Melkebeek, D.: Time-space tradeoffs for nondeterministic computation. In: *Proceedings of the 15th IEEE Conference on Computational Complexity*, pp. 2–13. IEEE, Los Alamitos (2000)
4. van Melkebeek, D.: A survey of lower bounds for satisfiability and related problems. *Foundations and Trends in Theoretical Computer Science* 2, 197–303 (2007)
5. Savitch, W.: Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences* 4, 177–192 (1970)
6. Williams, R.: Time-space tradeoffs for counting NP solutions modulo integers. *Computational Complexity* 17, 179–219 (2008)
7. Williams, R.: Alternation-trading proofs, linear programming, and lower bounds (manuscript, 2009)

Multiple Round Random Ball Placement: Power of Second Chance

Xiang-Yang Li^{1,*}, Yajun Wang², and Wangsen Feng³

¹ Department of Computer Science, Illinois Institute of Technology, USA
xli@cs.iit.edu

² Microsoft Research Asia, China
yajunw@microsoft.com

³ Department of Computer Science, Peking University, China
fengws@pku.edu.cn

Abstract. The traditional coupon collector’s problem studies the number of balls required to fill n bins if the balls are placed into bins uniformly at random. It is folklore that $\Theta(n \ln n)$ balls are required to fill the bins with high probability (*w.h.p.*).¹ In this paper, we study a variation of the random ball placement process. In each round, we assume the ability to acquire the set of empty bins after previous rounds and exclusively place balls into them uniformly at random. For such a k -round random ball placement process (k -RBP), we derive a sharp threshold of $n \ln^{[k]} n$ balls for filling n bins.²

We apply the bounds of k -RBP to the wireless sensor network deployment problem. Assume the communication range for the sensors is r and the deployment region is a 2D unit square. Let $n = (1/r)^2$. We show that the number of random nodes needed to achieve connectivity is $\Theta(n \ln \ln n)$ if we are given a “second chance” to deploy nodes, improving the previous $\Theta(n \ln n)$ bounds [8] in the one round case. More generally, under certain deployment assumption, if the random deployment in i -th round can utilize the information from the previous $i - 1$ rounds, the asymptotic number of nodes to satisfy connectivity is $\Theta(n \ln^{[k]} n)$ for k rounds. Similar results also hold if the sensing regions of the deployed nodes are required to cover the region of interest.

1 Introduction

In the coupon collector’s problem, there are n types of coupons. Each time, a coupon is chosen uniformly at random. The choices of coupons are mutually independent. We can view this problem as placing *identical and indistinguishable* balls into n distinguishable (numbered) bins. This problem has been extensively studied and well-understood [13]: when n is large, $n \ln n$ balls are sufficient and necessary to fill the bins *w.h.p.*.

* Xiang-Yang Li is partially supported by NSF CNS-0832120, National Natural Science Foundation of China under Grant No. 60828003, National Basic Research Program of China (973 Program) under grant No. 2006CB30300, the National High Tech. Research and Development Program of China (863 Program) under grant No. 2007AA01Z180, Hong Kong CERF under Grant PolyU-5232/07E, and Hong Kong RGC HKUST 6169/07.

¹ An event is said to happen *with high probability*, if it happens with probability at least $1 - 1/n$.

² Here $\ln^{[k]}$ denotes the natural logarithm iterated for k times.

In this paper, we employ the concept of “second chance” to design a multiple round random ball placement process. In our newly proposed ball placement process, we first randomly place a certain number of balls into n bins. We then acquire the set of empty bins after the first round and randomly and independently place balls into the current set of empty bins. In a general k round random ball placement process (k -RBP), before i th-round, we collect all empty bins after previous $i-1$ rounds, and place m_i balls randomly into the remaining empty bins. We are interested in the total number of balls, i.e., $\sum_{i=1}^k m_i$, required to fill all n bins *w.h.p.*. Our main result is a *sharp threshold* $n \ln^{[k]} n$ for the number of balls required to fill all bins for any k -RBP.

Our result is inspired by the “two choices” idea proposed in [15] for randomized load balancing. For each job to be scheduled, they randomly select d processors and assigns the job to the least loaded processor. Then the maximum load of all processors is, with high probability, at most $\frac{\log \log n}{\log d} + O(1)$ (for $d \geq 2$), instead of $\frac{\log n}{\log \log n}$ when $d = 1$. Thus giving each ball two choices leads to an exponential improvement in the maximum load. Our second chance approach is significantly different from the “two choices” idea. In our approach, the “choices” are temporal, why in [15], they must make the choices for a job instantaneously. Temporal variant random processes has appeared in studying *randomized rumor spreading* by Karp et. al. [10] implicitly, where they designed a two-phase “push” and “pull” algorithm.

Our work is motivated by the random deployment problem of wireless sensor networks. Gupta and Kumar [8] studied the critical transmission range r for the connectivity of the network formed by m random sensor nodes in a unit square. Their pioneering work showed that if $m\pi r^2 \geq c_1 \ln m$ for some constant $c_1 > 0$, then the graph (two nodes are connected iff their Euclidean distance is at most r) will be connected with probability 1 when $n \rightarrow \infty$. In addition, if $m\pi r^2 \leq c_0 \ln m$ for some special constant c_0 , the network will be disconnected with constant probability.

Using the result of k -RBP, we design a random deployment method employing the “second chance” concept. If we assume the ability to randomly deploy nodes in the “empty region” after previous rounds, $\Theta(n \ln^{[k]} n)$ nodes are needed to achieve connectivity and coverage, where k is the total number of rounds. In practice, though, it is hard to acquire “empty area” after previous rounds. Our solution is based on the possible “giant component” formed by the first round deployment. Indeed, if appropriate number of nodes are deployed in the first round, there exists a unique large connected component which touches the boundary with high probability. Localization algorithms [6, 5, 2, 17, 1] exist to locate the nodes in this component, which gives us a subset of the “empty area”. If we are able to randomly deploy wireless sensor nodes in the region outside of the largest connected component, a “second chance” deployment only needs $\Theta(n \ln \ln n)$ nodes to achieve connectivity and coverage both with high probability.

Our results, together with the power of multiple choices (by Mitzenmacher [15]) show that the system performances can be greatly improved if the choices (temporal choices used in this paper and spacial choices used in [15]) are only slightly relaxed. Besides the possible theoretical interest of k -RBP, the results developed here could be used to enhance our study of the performances of (random) wireless networks.

2 Multiple Round Random Ball Placement

In this section, we revisit the problem of randomly placing m *identical and indistinguishable* balls into n distinguishable (numbered) bins. Traditional coupon collector's problem studies the process that each ball is *independently* placed into bins uniformly at random. We refer this random process as *one round* random ball placement. We study the case that we are able to identify the empty bins after previous rounds and place balls *exclusively* into the remaining empty balls in the next round uniformly at random. Within each round, the placements of balls are *independent*. We are interested in the asymptotic number of balls required to fill all the bins with high probability.

Fact 1. $\forall n \geq 1$ and $|t| \leq n$, it holds that $e^t \left(1 - \frac{t^2}{n}\right) \leq \left(1 + \frac{t}{n}\right)^n \leq e^t$.

Definition 1 (k -Round Ball Placement (k -RBP)). In a k round ball placement, balls are randomly placed into n bins in k rounds. Let $n_0 = n$ be the original number of empty bins and n_i be the number of empty bins after i rounds. In the i th round, m_i balls are independently placed into n_{i-1} remaining empty balls uniformly at random.

Observe that a key requirement here for k -RBP is, after the first i rounds of random placement, we are able to determine the empty bins left and randomly place balls into the remaining empty balls. Denote by $Z_{m,n}$ the number of empty bins after m balls are placed randomly into n bins in *one round*. Then, $\mu = E[Z_{m,n}] = n(1 - 1/n)^m$. We define function $H(m, n, z) = \Pr[Z_{m,n} = z]$. Let $\exp(x) = e^x$ for any x . We have following occupancy bounds from [9].

Lemma 1 (Occupancy Bound 1 [9])

$$\forall \theta > 0, \Pr[|Z_{m,n} - \mu| \geq \theta\mu] \leq 2 \exp\left(-\frac{\theta^2 \mu^2 (n - 1/2)}{n^2 - \mu^2}\right)$$

Lemma 2 (Occupancy Bound 2 [9]). For $\theta > -1$,

$$H(m, n, (1 + \theta)\mu) \leq \exp(-((1 + \theta) \ln[1 + \theta] - \theta)\mu). \quad (1)$$

In particular, for $-1 \leq \theta < 0$, $H(m, n, (1 + \theta)\mu) \leq \exp(-\theta^2 \mu/2)$

We state the following requirements on k and n in this paper. *In the entire paper, all the results assume the following conditions.* Note that $k \geq 2$ is constant and all the conditions are satisfied when n is sufficient large (depending on k).

$$n \geq 2(\ln n)^2, \ln^{[k]} n \geq 2, \text{ and } \forall 1 \leq l \leq k, \ln^{[l-1]} n \geq 2 \ln 4 + 6 \ln^{[l]} n. \quad (2)$$

2.1 Upper Bound

We are interested in $m = \sum_i^k m_i$ so that with high probability there is no empty bin left, where m_i is the number of ball placed in i th round in a k -RBP. The following lemma discusses the case when $k = 1$, which can be proven by simple union bound.

Lemma 3. Let $\epsilon \in (0, 1)$ be a constant. If we randomly place $(1 + \epsilon)n \ln n$ balls into n bins, with probability at least $1 - \frac{1}{n^\epsilon}$, all the n bins are filled.

To simplify of presentation later, we define function

$$f(n, k, \delta) = \frac{1}{\ln^{[k]} n} \left(1 - \frac{\ln^{[k-1]} n \ln \delta}{n}\right).$$

Lemma 4. Let $\delta \in (0, 1)$ be a constant. If we randomly place $(1 + f(n, k, \delta))n \ln^{[k]} n$ balls into n bins, with probability at least $(1 - \delta)$, the number of empty bins left is at most $\frac{n}{\ln^{[k-1]} n}$.

Proof. Let $s = \ln^{[k-1]} n$ and then $\ln s = \ln^{[k]} n$. Consider the first n/s bins. The probability that these bins are empty is $(1 - 1/s)^{(1+f(n,k,\delta))n \ln s} \leq \exp(-\frac{(1+f(n,k,\delta))n \ln s}{s})$.

By union bound, the probability that there are more than $\frac{n}{s}$ empty bins is at most

$$\binom{n}{n/s} \exp(-\frac{(1+f(n,k,\delta))n \ln s}{s}) \leq \exp(-\frac{(1+f(n,k,\delta))n \ln s}{s})(e \cdot s)^{\frac{n}{s}}. \quad (3)$$

The inequality uses the Sterling's approximation $\forall 0 < m \leq n$, $(\frac{n}{m})^m \leq \binom{n}{m} \leq (\frac{ne}{m})^m$. Because $s^{1/s} = e^{\frac{\ln s}{s}}$, the last term in inequality (3) is δ . $\square$

Lemma 5. Let $2 \leq l \leq k-1$. Randomly place $2n$ balls in $\frac{n}{\ln^{[l]} n}$ bins. With probability at least $(1 - \frac{1}{kn})$, the number of empty bins left is at most $\frac{n}{\ln^{[l-1]} n}$.

Proof. The probability p that the first $\frac{n}{\ln^{[l-1]} n}$ bins are empty is $p = \left(1 - \frac{\ln^{[l]} n}{\ln^{[l-1]} n}\right)^{2n} \leq \exp(-\frac{2n \ln^{[l]} n}{\ln^{[l-1]} n})$. By union bounds, the probability that there are more than $\frac{n}{\ln^{[l-1]} n}$ empty bins is at most

$$\begin{aligned} p \cdot \binom{\frac{n}{\ln^{[l]} n}}{\frac{n}{\ln^{[l-1]} n}} &\leq p \left(\frac{e \ln^{[l-1]} n}{\ln^{[l]} n} \right)^{\frac{n}{\ln^{[l-1]} n}} \leq p \cdot e^{\frac{n}{\ln^{[l-1]} n}} (\ln^{[l-1]} n)^{\frac{n}{\ln^{[l-1]} n}} \\ &\leq \exp(-\frac{2n \ln^{[l]} n}{\ln^{[l-1]} n} + \frac{n}{\ln^{[l-1]} n} + \frac{n \ln^{[l]} n}{\ln^{[l-1]} n}) \leq \exp(-2 \ln n) \leq \frac{1}{kn}. \end{aligned}$$

The last inequality comes from Eq. (2). $\square$

Theorem 1. There exists a k -RBP for n bins with $(1 + f(n, k, \frac{1}{kn}) + \frac{2(k-1)}{\ln^{[k]} n})n \ln^{[k]} n$ balls, such that it is sufficient to fill all bins with probability at least $(1 - 1/n)$.

Proof. We first place $(1 + f(n, k, \frac{1}{kn}))n \ln^{[k]} n$ balls randomly into n empty bins. From Lemma 4, with probability at least $(1 - \frac{1}{kn})$, the number of empty bins after first round is at most $\frac{n}{\ln^{[k-1]} n}$.

In i th round, for $2 \leq i \leq k-1$, we place $2n$ balls randomly into the empty bins after $(i-1)$ th round. By Lemma 5, after $k-1$ rounds, with probability at least $(1 - \frac{1}{kn})^{k-1} \geq 1 - \frac{k-1}{kn}$, the number of empty bins left is at most $n_{k-1} \leq \frac{n}{\ln n}$.

In the k -th round, we randomly place another $2n$ balls. From Lemma 3, in order to achieve probability at least $(1 - \frac{1}{kn})$, it requires $(1 + \frac{\ln(kn)}{\ln n_{k-1}})n_{k-1} \ln n_{k-1} \leq 2n$ balls (when $n_{k-1} \leq \frac{n}{\ln n}$ and $k \leq \ln n$). Therefore, randomly placing $2n$ balls suffices to fill all remaining empty balls with probability at least $(1 - \frac{1}{kn})$.

In total, we place $(1 + f(n, k, \frac{1}{kn}) + \frac{2(k-1)}{\ln^{[k]} n})n \ln^{[k]} n$ balls. The probability of success is at least $(1 - \frac{1}{k \cdot n})^k \geq 1 - 1/n$. $\square$

We remark that $f(n, k, \frac{1}{kn}) + \frac{2(k-1)}{\ln^k n} = o(1)$ when $n \rightarrow \infty$ and k remains constant. Essentially, $n \ln^{[k]} n$ balls is sufficient to fill n bins with high probability by a k -RBP.

2.2 Lower Bound

We now show that the bound of $n \ln^{[k]} n$ in Theorem 1 is tight. In particular, if the number of balls is $(1 - o(1))n \ln^{[k]} n$, any k -RBP will always have empty bins left with constant probability. The case for $k = 1$ is simple and its proof is omitted.

Lemma 6. *Randomly place $n \ln n$ balls into n bins. With probability at least $1 - e^{1/4}$, there is at least one empty bin.*

Compared with the upper bound, the difficulty for the lower bound is that we do not have any constraint on the distribution of m balls over k rounds of the deployment. We essentially have to show that, regardless of the choices of m_i ($1 \leq i \leq k$), there are empty bins with constant probability when $m = \sum_i m_i$ is less than some number.

Lemma 7. *Randomly place $(1 - \frac{4 \ln^{[k+1]} n}{\ln^{[k]} n})n \ln^{[k]} n$ balls into n bins. With probability $1 - 2/e$, there are at least $\frac{n(\ln^{[k]} n)^2}{\ln^{[k-1]} n}$ empty bins left.*

Proof. Let $\epsilon = \frac{4 \ln^{[k+1]} n}{\ln^{[k]} n}$. Let Z be the random variable for the number of empty bins. We have $\mu = E[Z] = n(1 - \frac{1}{n})^{(1-\epsilon)n \ln^{[k]} n}$.

By occupancy bound 1, $\Pr[Z \geq \frac{\mu}{2}] \geq 1 - \Pr[|Z - \mu| \geq \frac{\mu}{2}] \geq 1 - 2e^{-\frac{\mu^2(n-1/2)}{n^2 - \mu^2}}$. Since $n > 2 \ln n$ by Eqn. (2), $1 - \frac{(1-\epsilon) \ln^{[k]} n}{n} \geq 1/2$. By Fact 1, $\mu \geq \frac{n}{2} e^{-(1-\epsilon) \ln^{[k]} n}$. As $\ln^{[k]} > 2$, we also have $e^{-(1-\epsilon) \ln^{[k]} n} = \frac{(\ln^{[k]} n)^4}{\ln^{[k-1]} n} \geq \frac{4(\ln^{[k]} n)^2}{\ln^{[k-1]} n}$. Hence, $\mu \geq \frac{2n(\ln^{[k]} n)^2}{\ln^{[k-1]} n}$. As $n \geq (\ln n)^2$, we have $\mu^2 \geq n$. Hence, $\Pr[Z \geq \frac{\mu}{2}] \geq 1 - 2e^{-\frac{\mu^2}{n}} \geq 1 - 2/e$. $\square$

Lemma 8. *Let $2 \leq l \leq k$. Randomly place $n \ln^{[k]} n$ balls into $\frac{n(\ln^{[l]} n)^2}{\ln^{[l-1]} n}$ bins in one-round. With probability $1 - 2/e$, the number of remaining empty bins left is at least $\frac{n(\ln^{[l-1]} n)^2}{\ln^{[l-2]} n}$.*

Proof. Since $\ln^{[k]} n \leq \ln^{[l]} n$ when $l \leq k$, we can assume the number of balls placed is $n \ln^{[l]} n$ instead. Let Z be the random variable for the number of empty bins. The expected number of empty bins is $\mu = E[Z] = \frac{n(\ln^{[l]} n)^2}{\ln^{[l-1]} n} (1 - \frac{\ln^{[l-1]} n}{n(\ln^{[l]} n)^2})^{n \ln^{[l]} n}$.

Let $t = \frac{\ln^{[l-1]} n}{\ln^{[l]} n}$. By Fact 1, we have $\mu \geq \frac{n(\ln^{[l]} n)^2}{\ln^{[l-1]} n} (1 - \frac{t^2}{n \ln^{[l]} n}) e^{-t}$. As $n \geq 2(\ln^{[l-1]} n)^2$, it implies that $(1 - \frac{t^2}{n \ln^{[l]} n}) \geq 1/2$.

As $\ln^{[l]} n \geq 2$, $\mu \geq \frac{n(\ln^{[l]} n)^2}{2 \ln^{[l-1]} n} (\ln^{[l-2]} n)^{-1/2}$. It is straightforward to verify that $\mu \geq \frac{2n(\ln^{[l-1]} n)^2}{\ln^{[l-2]} n}$ by Eq. (2). Now let $m = \frac{n(\ln^{[l]} n)^2}{\ln^{[l-1]} n}$. From occupancy bound 1, $\Pr[Z \geq \frac{\mu}{2}] \geq 1 - \Pr[|Z - \mu| \geq \frac{\mu}{2}] \geq 1 - 2e^{-\frac{\mu^2(m-1/2)}{m^2-\mu^2}}$. Since $n \geq 2(\ln n)^2$, $\mu^2 \geq m$. $\Pr[Z \geq \frac{\mu}{2}] \geq 1 - 2e^{-\frac{\mu^2}{m}} \geq 1 - 2/e$. $\square$

Theorem 2. *Randomly place $(1 - \frac{4 \ln^{[k+1]} n}{\ln^{[k]} n})n \ln^{[k]} n$ balls into n bins by any k -RBP. With constant probability, there exists at least one empty bin.*

Proof. Let $m = (1 - \frac{4 \ln^{[k+1]} n}{\ln^{[k]} n})n \ln^{[k]} n$. Based on Lemma 7 and Lemma 8, before last round, the number of empty bins is at least $\frac{n(\ln \ln n)^2}{\ln n}$ with constant probability, if m balls are placed in each round of the first $k - 1$ rounds.

Let $m_k = \frac{n(\ln \ln n)^2}{\ln n}$. As $n \geq 2 \ln n$, we have $\ln n \geq 2 \ln \ln n$. Hence $m_k \ln m_k \geq n(\ln \ln n)^2/2 \geq n \ln^{[k]} n$. By Lemma 6, $n \ln^{[k]} n$ balls will leave empty bins with constant probability after k -th round. Thus, with constant probability, any k -RBP with m balls will have empty bins $\square$

2.3 Unit Bins in One Round

We denote *unit bins* as the bins that contain exactly one ball. As shown later, the existence of unit bins is closely related to connectivity of the randomly deployed network. We develop the bound on the number of unit bins similar to the occupancy bound 1 [9, 4]. We follow the proof scheme of [4] with the “bounded difference” method.

Let $Z_i (1 \leq i \leq n)$ be the indicating random variable which is 1 if the i th bin is a *unit bin* and 0 otherwise. Let $Z = \sum_{i=1}^n Z_i$ be the number of unit bins. We are interested in tail bounds on the distribution of Z . The following lemma is from McDiarmid [14].

Lemma 9 (McDiarmid [14]). *Let $[n] = \{1, 2, \dots, n\}$. Let $X_1, \dots, X_n$ be independent random variables, variable X_i taking values in a finite set A_i for each $i \in [n]$, and suppose the function f satisfies the following “bounded difference” conditions: for each $i \in [n]$, there is a constant c_i such that for any $x_k \in A_k, k \in [i - 1]$ and for $x_i, x'_i \in A_i$*

$$\begin{aligned} |E[f(\mathbf{X})|X_1 = x_1, \dots, X_{i-1} = x_{i-1}, X_i = x_i] \\ - E[f(\mathbf{X})|X_1 = x_1, \dots, X_{i-1} = x_{i-1}, X_i = x'_i]| \leq c_i, \end{aligned}$$

then $\Pr[|f(\mathbf{X}) - E[f(\mathbf{X})]| > t] < 2 \exp\left(-\frac{t^2}{2 \sum_i c_i^2}\right)$.

Theorem 3 (Occupancy bound for unit bins). *Randomly place m balls into n bins. Let Z be the number of unit bins left. For any $\theta > 0$,*

$$\Pr[|Z - \mu| \geq \theta \mu] \leq 2 \exp\left(-\frac{\theta^2 \mu^2 (2n - 1)}{2(2n + m)^2 (1 - (1 - \frac{1}{n})^{2m})}\right)$$

where $\mu = E[Z] = m(1 - \frac{1}{n})^{m-1}$ is the expected number of unit bins.

Proof. We view the variable Z as $Z = Z(B_1, \dots, B_m)$ where the random variable B_k takes values in the set $[n]$ indicating which bin the ball k occupies, for $k \in [m]$. Define $B = \{B_1 = b_1\} \wedge \dots \wedge \{B_i = b_i\}$ and $B' = \{B_1 = b_1\} \wedge \dots \wedge \{B_{i-1} = b_{i-1}\} \wedge \{B_i = b'_i\}$. To apply Lemma 9, we need to compute the difference $D = |E[Z|B] - E[Z|B']|$.

Note that for any $j \in [n]$ and $j \neq b_i, b'_i$, $E[Z_j|B] = E[Z_j|B']$. Thus, $D = |E[Z_{b_i} + Z_{b'_i}|B] - E[Z_{b_i} + Z_{b'_i}|B']|$. Clearly, we are interested in the case $b = b_i \neq b'_i = b'$. Let $I = \{b_1, \dots, b_{i-1}\}$. As expectations are non-negative and by symmetry, $D \leq E[Z_b + Z_{b'}|B]$. Let $n(b')$ be the number of times b' appears in I , e.g., the number of balls in b' th bin after the first i balls are randomly placed.

$$E[Z_b|B] = \begin{cases} 0 & b \in I \\ (1 - \frac{1}{n})^{m-i} & \text{otherwise} \end{cases} \quad E[Z_{b'}|B] = \begin{cases} 0 & n(b') \geq 2 \\ (1 - \frac{1}{n})^{m-i} & n(b') = 1 \\ \frac{m-i}{n}(1 - \frac{1}{n})^{m-i-1} & \text{otherwise} \end{cases}$$

Hence $E[Z_b + Z_{b'}|B] \leq 2(1 - \frac{1}{n})^{m-i} + \frac{m-i}{n}(1 - \frac{1}{n})^{m-i-1}$. We have

$$D \leq 2(1 - \frac{1}{n})^{m-i} + \frac{m-i}{n}(1 - \frac{1}{n})^{m-i-1} \leq \frac{(2n+m)}{n}(1 - \frac{1}{n})^{m-i} \quad (4)$$

As a result, for $i \in [m]$, $D = |E[Z|B] - E[Z|B']| \leq \frac{(2n+m)}{n}(1 - \frac{1}{n})^{m-i} = c_i$.

$$\sum_{i=1}^m c_i^2 = \frac{(2n+m)^2}{n^2} \sum_{i=1}^m [(1 - \frac{1}{n})^{m-i}]^2 = \frac{(2n+m)^2(1 - (1 - \frac{1}{n})^{2m})}{2n-1} \quad (5)$$

The theorem follows directly by applying Lemma 9. $\square$

Theorem 4. Let $n \geq (12)^4$. Randomly place m balls into n bins, where $1 < m < n \ln n/4$. With constant probability, there exists one unit bin afterwards.

Proof. Let Z be random variable of the number of unit bins. Then $E[Z] = \mu = m(1 - \frac{1}{n})^{m-1}$, and $E[Z^2] = m^2(1 - \frac{1}{n})^{m-1}$.

First we consider the case that $m \leq n/2$. By the *second moment method* $\Pr[Z = 0] \leq \frac{E[Z^2]}{(E[Z])^2} - 1 = 1/(1 - \frac{1}{n})^{m-1} - 1$. Note that by Fact 1, $(1 - \frac{1}{n})^{m-1} \geq (1 - \frac{1}{n})^m \geq (1 - \frac{m}{n^2})e^{-\frac{m}{n}}$. Because $m \leq n^2/10$, $(1 - \frac{1}{n})^m \geq \frac{9}{10}e^{-1/2}$, $\Pr[Z = 0] \leq \frac{10e^{1/2}}{9} - 1 < 1$.

Second, consider the case that $m = cn$ for $c \in (1/2, \ln n/4)$. By Theorem 3,

$$\begin{aligned} \Pr[Z = 0] &\leq 2 \exp\left(-\frac{\mu^2(2n-1)}{8(n+m)^2(1 - (1 - \frac{1}{n})^{2m})}\right) \leq 2 \exp\left(-\frac{\mu^2 n}{8(n+m)^2}\right) \\ &\leq 2 \exp\left(-\frac{m^2(1 - \frac{1}{n})^{2m-2}n}{8(n+m)^2}\right) \leq 2 \exp\left(-\frac{(1 - \frac{1}{n})^{2m}n}{8(1 + \frac{n}{m})^2}\right). \end{aligned}$$

Because $(1 - \frac{1}{n})^{2m} \geq (1 - \frac{2m}{n^2})e^{-2m/n} \geq \frac{1}{2\sqrt{n}}$ and $(1 + \frac{n}{m})^2 \leq 9$. Hence $\Pr[Z = 0] \leq 2 \exp(-\frac{\sqrt{n}}{144}) \leq 2/e < 1$. $\square$

3 Application for Wireless Sensor Network Deployment

Largest Connected Component. We assume the communication range for all sensor nodes is r and the deployment region is a 2D unit square. Let $n = (1/r)^2$. Carruthers and King [3] proved that there exists a unique large connected component which covers at least a constant portion of the square in expectation, if the number of random nodes deployed is $\Theta(n)$. We show that, by increasing the number of nodes slightly, the portion of the area covered by the largest connected component will be $1 - o(1)$. The proofs are omitted due to space limitation.

Lemma 10. *Let $n \geq 4(\ln n)^5$. Randomly place $2n \ln \ln n - n \ln 2$ balls into n bins. With probability at least $1 - \frac{1}{2n}$, the number of empty bins is at most $\frac{4n}{(\ln n)^2}$.*

Together with the *empty path length lemma* [3], we have the following result.

Theorem 5 (Largest Connected Component). *Assume $n = (1/r)^2 \geq 36$ by randomly placing $36n \ln \ln(36n)$ wireless nodes in the square, the largest connected component covers area with size more than $1 - \frac{2}{\ln(36n)}$ with probability at least $1 - 1/n$.*

Coverage. In traditional one round random deployment, it is shown that we need $\Theta(n \ln n)$ nodes [19] to provide full coverage with high probability. We first assume we can deploy nodes outside the region covered by wireless nodes deployed in previous rounds. Based on the technical results derived in Section 2, we obtain tight bounds for any “second chance” deployment strategy in k rounds. Then, we study the case that we are only able to deploy nodes outside the largest connected component from the first round.

Let $r' = 1/\lceil \sqrt{2}/r \rceil \in [\sqrt{2}r/4, \sqrt{2}r/2]$. We divide the unit square into cells with side length r' . The number of cells is still $O(n)$. By Theorem 1, $O(n \ln^{[k]} n)$ nodes sufficiently fill all the cells with probability at least $1 - 1/n$ in k rounds, which assures the coverage. If we set the cells' side length to $r'' = 2/\lfloor 1/r \rfloor \in [2r, 4r]$, we need at least one node for each such cell to assure full coverage. Since the number of cells is $O(n)$, by Theorem 2, we need $\Omega(n \ln^{[k]} n)$ wireless nodes to assure full coverage.

Theorem 6. *Assume we can randomly deploy wireless nodes outside the covered region in previous rounds. With $\Theta(n \ln^{[k]} n)$ wireless nodes, we can cover the square in k rounds with probability at least $1 - 1/n$.*

The assumption that we can determine the empty cells from previous rounds is sometimes too restrict. On the other hand, by Theorem 5, after one round with $O(n \ln \ln n)$ nodes, there is a large connected component, which will cover at least one point on the boundary with high probability [3]. Therefore we can easily probe this largest connected component by simply querying the sensors along the region boundary. If we assume we can randomly place nodes outside this largest connected component, $\Theta(n \ln \ln n)$ nodes suffice to cover the square in two rounds.

Theorem 7. *Assume we can randomly deploy wireless nodes outside the area covered by the largest connected component in the first round. With $\Theta(n \ln \ln n)$ wireless nodes, we can cover the square region by a “second chance” random deployment with probability at least $1 - 1/n$.*

Connectivity. The asymptotic upper bound on the number of nodes needed for achieving network connectivity is the same with the coverage. In fact, if a set of nodes achieves coverage for a square region with communication range $r/2$, they form a connected network under communication range r . Hence $O(n \ln^{[k]} n)$ nodes is sufficient to assure connectivity by a “second chance” random deployment in k rounds and $O(n \ln \ln n)$ in two rounds. The lower bound, however, requires an argument that is different with the coverage case. We first exclude the trivial case when we only deploy one node.

Let $r' = \frac{1}{\lfloor 1/5r \rfloor}$ and $n' = (1/r')^2 = O(n)$. We first divide the unit square into grid cells with side-length r' . Note that $r' \geq 5r$. If we deploy $\Theta(n \ln \ln n)$ nodes in the first round, by Lemma 7 the number of empty cells after this round is at least $\frac{n'(\ln \ln n')^2}{\ln n'}$ with a probability at least $1 - 2/e$. Consider the second round of the deployment. If we randomly deploy another $\Theta(n \ln \ln n)$ balls in the second round, by Theorem 4, with constant probability, there exists a *unit cell* afterwards.

Consider one of the unit bins left, with constant probability, the node inside will be in the middle, i.e., the center $r \times r$ square. Since the side length of the cell is $5r$, this node will be disconnected.

Theorem 8. *Assume we can randomly deploy wireless nodes outside the region covered by the nodes in the first round. With $\Theta(n \ln \ln n)$ nodes, the network is connected with high probability using a “second chance” random deployment in two rounds.*

4 Conclusions

In this paper, we propose a multiple round random ball placement process k -RBP. The asymptotic number of balls required to fill n bins is $\Theta(n \ln^{[k]} n)$, instead of the $O(n \ln n)$ in the original coupon collector’s problem. In general, we expect this result to find applications in other problems.

By applying k -RBP, our “second chance” random deployment for wireless sensor networks significantly reduces the number of nodes required to achieve connectivity and coverage compared to traditional one round random deployment. This result can be viewed as a first step towards a more complete study of trade-off between the network quality and the deployment complexity. There are a number of interesting and challenging problems regarding the network formed by the “second chance” deployment, such as critical transmission range and network capacities.

References

1. Basu, A., Gao, J., Mitchell, J.S.B., Sabhnani, G.: Distributed localization using noisy distance and angle information. In: ACM MobiHoc, pp. 262–273 (2006)
2. Biswas, P., Ye, Y.: Semidefinite programming for ad hoc wireless sensor network localization. In: Zhao, F., Guibas, L. (eds.) Proc. of Third International Workshop on Information Processing in Sensor Networks (2004)
3. Carruthers, S., King, V.: Connectivity of Wireless Sensor Networks with Constant Density. In: Nikolaidis, I., Barbeau, M., Kranakis, E. (eds.) ADHOC-NOW 2004. LNCS, vol. 3158, pp. 149–157. Springer, Heidelberg (2004)

4. Dubhashi, D.: Simple proofs of occupancy tail bounds. *Random Structure Algorithms* 11(2), 119–123 (1997)
5. Eren, T., Goldenberg, D., Whitley, W., Yang, Y., Morse, S., Anderson, B., Belhumeur, P.: Rigidity, computation, and randomization of network localization. In: *Proc. of IEEE INFOCOM 2004* (2004)
6. Goldenberg, D.K., Bihler, P., Yang, Y.R., Cao, M., Fang, J., Morse, A.S., Anderson, B.D.O.: Localization in sparse networks using sweeps. In: *ACM MobiCom*, pp. 110–121 (2006)
7. Gupta, P., Kumar, P.: Capacity of wireless networks. *IEEE Transactions on Information Theory* IT-46, 388–404 (1999)
8. Gupta, P., Kumar, P.R.: Critical power for asymptotic connectivity in wireless networks. In: Fleming, W.H., McEneaney, W.M., Yin, G., Zhang, Q. (eds.) *Stochastic Analysis, Control, Optimization and Applications* (1998)
9. Kamath, A., Motwani, R., Palem, K., Spirakis, P.: Tail bounds for occupancy and the satisfiability threshold conjecture. *Random Structures and Algorithms* 7(1) (1995)
10. Karp, R., Schindelhauer, C., Shenker, S., Vocking, B.: Randomized rumor spreading. In: *IEEE Foundation of Computer Science (FOCS)* (2000)
11. Kyasanur, P., Vaidya, N.H.: Capacity of multi-channel wireless networks: impact of number of channels and interfaces. In: *ACM MobiCom*, pp. 43–57 (2005)
12. Li, X.-Y., Tang, S.-J., Ophir, F.: Multicast capacity for large scale wireless ad hoc networks. In: *ACM Mobicom*, pp. 266–277 (2007)
13. Motwani, R., Raghavan, P.: *Randomized Algorithms*. Cambridge Uni. Press, Cambridge (1995)
14. McDiarmid, C.: On the method of bounded differences. *Surveys in Combinatorics* 141, 148–188 (1989)
15. Mitzenmacher, M.: The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.* 12(10), 1094–1104 (2001)
16. Shakkottai, S., Liu, X., Srikant, R.: The multicast capacity of ad hoc networks. In: *ACM MobiHoc* (2007)
17. So, A.M.-C., Ye, Y.: Theory of semidefinite programming for sensor network localization. In: *Proc. 16th ACM-SIAM Symp. on Discrete Algorithms*, pp. 405–414 (2005)
18. Wan, P.-J., Yi, C.-W.: Asymptotic critical transmission radius and critical neighbor number for k -connectivity in wireless ad hoc networks. In: *ACM MobiHoc*, pp. 1–8 (2004)
19. Wan, P.-J., Yi, C.-W.: Coverage by randomly deployed wireless sensor networks. *IEEE/ACM Transactions on Networking* 14, 2658–2669 (2006)
20. Xue, F., Kumar, P.R.: On the θ -coverage and connectivity of large random networks. *IEEE/ACM Transactions on Networking* 14, 2289–2299 (2006)

The Weighted Coupon Collector's Problem and Applications

Petra Berenbrink¹ and Thomas Sauerwald²

¹ Simon Fraser University, School of Computing Science,
8888 University Drive, Burnaby B.C. V5A 1S6

² International Computer Science Institute, Berkeley, CA, USA

Abstract. In the classical coupon collector's problem n coupons are given. In every step one of the n coupons is drawn uniformly at random (with replacement) and the goal is to obtain a copy of all the coupons. It is a well-known fact that in expectation $n \sum_{k=1}^n 1/k \approx n \ln n$ steps are needed to obtain all coupons.

In this paper we show two results. First we revisit the weighted coupon collector case where in each step every coupon i is drawn with probability p_i . Let $\mathbf{p} = (p_1, \dots, p_n)$. In this setting exact but complicated bounds are known for $\mathbf{E}[\mathcal{C}(\mathbf{p})]$, which is the expected time to obtain all n coupons. Here we suggest the following rather simple way to approximate $\mathbf{E}[\mathcal{C}(\mathbf{p})]$. Assume $p_1 \leq p_2 \leq \dots \leq p_n$ and take $\sum_{i=1}^n 1/(ip_i)$ as an approximation. We prove that, rather unexpectedly, this expression approximates $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ by a factor of $\Theta(\log \log n)$. We also present an extension that achieves an approximation factor of $\mathcal{O}(\log \log \log n)$.

In the second part of the paper we derive some combinatorial properties of the coupon collecting processes. We apply these properties to show results for the following simple randomized broadcast algorithm. A graph G is given and one node is initially informed. In each round, every informed node chooses a random neighbor and informs it. We restrict G to the class of trees and we show that the expected broadcast time is *maximized* if and only if G is the star graph. Besides being the first rigorous extremal result, our finding nicely contrasts with a previous result by Brightwell and Winkler [2] showing that for the star graph the cover time of a random walk is *minimized* among all trees.

1 Introduction

In the standard *coupon collector's problem* n coupons are given and in every step of the process one of these coupons is chosen with the same probability uniformly at random, with replacement. It is well-known that $\mathbf{E}[\mathcal{C}]$, the expected time to draw all coupons is $\sum_{k=1}^n n/(n-k+1) = n \cdot \sum_{k=1}^n 1/k \approx n \ln n$. In this paper we consider two closely related problems. Firstly, we consider the *weighted version* of the coupon collector's problem which is defined as follows. Again, n is the number of coupons. For each coupon $i \in \{1, \dots, n\}$, let $0 < p_i < 1$ be the probability of choosing the i th coupon in every step. Let $\mathbf{p} = (p_1, \dots, p_n)$ a given probability vector and assume that $p_1 \leq p_2 \leq \dots \leq p_n$. We are interested in simple bounds on $\mathbf{E}[\mathcal{C}(\mathbf{p})]$, the expected time to draw all coupons. With a simple approximation we mean an approximation that is (i) similar to the bound for the standard unweighted case, and (ii) a bound which can be easily

evaluated, in contrast to a bound based on the inclusion-exclusion principle. Note that the latter bound has exponentially many terms.

The second problem we investigate is random broadcasting on graphs. The broadcast algorithm we assume is known as Push algorithm [4] or Randomized Rumor Spreading [10]. The algorithm is defined as follows. Let $G = (V, E)$ be an undirected, simple and connected graph with n vertices. Initially, there is a single *informed* vertex $s \in V$ which owns a piece of information r . The goal is to inform all other vertices (i.e. sent a copy of r to all other vertices). The time is divided in steps and in every step every informed vertex chooses a random neighbor and sends a copy of r to it. Our goal is to find the graph so that the expected number of steps to inform all nodes is maximized.

To see that the coupon collector's problem and the broadcast algorithm described above are closely related let us consider the star network. The network has node v in the middle and n outer nodes which are only connected to v . If in the beginning node v is informed, then the broadcast algorithm is equivalent to the standard coupons collector's problem (with $n - 1$ coupons). Sending a message to a neighbour w is nothing else than drawing coupon w . Now suppose that $G = (V, E)$ is a tree with n vertices and assume that in the beginning the root of the tree is informed. Then the broadcast algorithm can be regarded as a parallelized version of the coupons collector's problem where all informed nodes draw coupons at the same time, but they are only allowed to draw coupons corresponding to their neighbors.

1.1 Known Results

Coupon Collector. There are many results for this well-studied problem and certain variations of it (e.g., [1]). Here we only review results which are concerned with the estimation of $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ for the weighted case. The exact value of $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ can be calculated by relating the coupon-collecting-process to independent poisson processes (see [12, p. 300] or [9]):

$$\mathbf{E}[\mathcal{C}(\mathbf{p})] = \int_0^\infty \left(1 - \prod_{i=1}^n (1 - \exp(-p_i \cdot x)) \right) dx.$$

Similar formulas can be found in [9] for the expected time to obtain k different coupons, where $k \leq n$. Another exact bound that uses the inclusion-exclusion principle can be found in [13, p. 386].

$$\mathbf{E}[\mathcal{C}(\mathbf{p})] = \sum_j \frac{1}{p_j} - \sum_{j < k} \frac{1}{p_j + p_k} + \sum_{i < j < k} \frac{1}{p_i + p_j + p_k} + \dots + (-1)^{n+1} \cdot \frac{1}{\sum_i p_i}.$$

Computing this sum is rather prohibitive, since it involves exponentially many summands.

We now recall three basic bounds on $\mathbf{E}[\mathcal{C}(\mathbf{p})]$. In [11, p. 414] it is shown that

$$\mathbf{E}[\mathcal{C}(\mathbf{p})] \leq \mathcal{U}_2(\mathbf{p}) := \sum_{i=1}^n \frac{1}{i \cdot p_i}.$$

From [11, p. 414] we get

$$\mathcal{L}_1(\mathbf{p}) := \sum_{i=1}^n \frac{1}{(n-i+1) \cdot p_i} \leq \mathbf{E}[\mathcal{C}(\mathbf{p})],$$

and from [11, p. 452]

$$\mathcal{L}_2(\mathbf{p}) := \frac{\mathcal{H}_n}{n} \cdot \sum_{i=1}^n \frac{1}{p_i} \leq \mathbf{E}[\mathcal{C}(\mathbf{p})].$$

All these bound are given without any estimations of their approximation ratios.

Randomized Broadcast. As mentioned earlier, the coupon-collecting-process is closely related to the broadcast algorithm where every node is allowed to communicate with a randomly chosen neighbour in every step. This algorithm was shown to broadcast in $\Theta(\log n)$ steps on hypercubes and random graphs (including complete graphs) by Feige et al. [8]. Moreover, [8] proved an upper bound of $12n \ln n$ on the broadcast time for general graphs. The latter result was improved to $(1 + o(1)) \cdot n \ln n$ in [5]. Note that this result is tight up to a factor of $(1 + o(1))$, since the broadcast time in the star graph with $n + 1$ nodes is bounded by $\mathbf{E}[\mathcal{C}]$. However, the question which families of graphs *exactly* maximizes (or minimizes) the expected broadcast time remained open.

1.2 Our Contribution

Let us summarize all bounds considered in this paper (and their approximation ratios) on the weighted coupon collector's problem. For any integer $k \in \mathbb{N}$, let $\mathcal{H}_k$ be the k -th harmonic number. For readability, we use $\mathcal{L}$ for lower bounds and $\mathcal{U}$ for upper bounds. In the following we define the *approximation ratio* of the lower bound $\mathcal{L}(\mathbf{p})$ as $\mathbf{E}[\mathcal{C}(\mathbf{p})] / \mathcal{L}(\mathbf{p})$, and that of the upper bound $\mathcal{U}(\mathbf{p})$ as $\mathcal{U}(\mathbf{p}) / \mathbf{E}[\mathcal{C}(\mathbf{p})]$. Note that every upper bound (lower bound) can be transformed into a lower bound (upper bound) by a division (multiplication) with the approximation factor.

Our contribution is a rigorous analysis of the approximations ratios. While we prove that $\mathcal{U}_2(\mathbf{p})$ and $\mathcal{U}_3(\mathbf{p})$ approximate $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ up to $\Theta(\log \log n)$, we also present an

Approximation	Approximation-Ratio	Reference
$\mathcal{L}_1(\mathbf{p}) = \sum_{i=1}^n 1/((n-i+1) \cdot p_i)$	$\Omega(n \log n)$ (Lemma 3.1)	[11, p. 414]
$\mathcal{L}_2(\mathbf{p}) = \mathcal{H}_n \cdot \sum_{i=1}^n 1/(p_i \cdot n)$	$\Omega(n \log n)$ (Lemma 3.1)	[11, p. 452]
$\mathcal{U}_1(\mathbf{p}) = \ln n / p_1$	$\Theta(\log n)$ (Lemma 3.2)	Folklore
$\mathcal{U}_2(\mathbf{p}) = \sum_{i=1}^n 1/(i \cdot p_i)$	$\Theta(\log \log n)$ (Thms. 3.4 and 3.5)	[11, p. 414]
$\mathcal{U}_3(\mathbf{p}) = \sum_{i=1}^n 1/(\sum_{k=1}^i (1/p_k))$	$\Theta(\log \log n)$ (Thms. 3.4 and 3.5)	This paper
$\mathcal{U}_4(\mathbf{p}) = \sum_{i=1}^{\ln \ln n} (1/i) \cdot (1/(e^{j_i-1} p_{j_i})) \cdot \mathcal{H}_{g_{j_i}}$	$\mathcal{O}(\log \log \log n)$ (Theorem 3.6)	This paper

Fig. 1. Summary of various approximations on $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ and their approximation ratios. All approximations require that the vector $\mathbf{p}$ is ordered increasingly. The exact definition of $\mathcal{U}_4(\mathbf{p})$ can be found in Theorem 3.6.

extension with an approximation ratio $\mathcal{O}(\log \log \log n)$. These ratios are much better than the polynomial approximation ratios of $\mathcal{L}_1(\mathbf{p})$ and $\mathcal{L}_2(\mathbf{p})$, though at a first glance, these approximation formulas look somewhat similar.

In the second part of the paper, we derive bounds for $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ on slight variations of the weighted coupon collector's problem. We apply these bounds inductively to prove that the star graph is the tree which maximizes the expected broadcast time.

It is well-known that the cover time¹ of a random walk is *asymptotically* minimized by the complete graph, and the cover time is *asymptotically* maximized by the lollipop graph [7,6] (the lollipop graph is a complete graph with $n/3$ vertices attached by a path of length $2n/3$). However, it is known that the complete graph is *not* the graph with the *minimum* cover time and the question whether the lollipop graph has the *maximal* cover time remains unsolved. Only for the subclass for trees Brightwell and Winkler [2] were able to show that the star graph maximizes the cover time.

Intuitively, one difficulty in proving rigorous extremal results for the broadcast and the cover time of random walks is that both the cover- and broadcast time are not monotone under adding edges (cf. [3]). For example, assume we start with the line graph (with cover time n^2), then by adding edges we transform the graph into a lollipop graph (with a cover time n^3). By adding further edges we get the complete graph (with a cover time $n \ln n$). With other, but straightforward examples one can also prove the corresponding result for the broadcast time.

2 Notation and Definitions

2.1 Weighted Coupon Collector

Let us introduce the weighted coupon collector's problem (more) formally. We are given a probability vector $\mathbf{p} = (p_1, \dots, p_n)$ with $p_1 \leq p_2 \leq \dots \leq p_n$. In each step, coupon i is drawn uniformly at random with probability p_i . We assume that $0 \leq \sum_{i=1}^n p_i \leq 1$; note that if $\sum_{i=1}^n p_i < 1$, then with probability $1 - \sum_{i=1}^n p_i$ no coupon will be drawn in each round. The random variable $\mathcal{C}(\mathbf{p})$ is the time to draw all n coupons and $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ is its expectation.

Observation 2.1. *Let $\mathbf{p}$ and $\mathbf{q}$ be the two probability vectors such that $p_i \geq q_i$ for all $i \in \{1, \dots, n\}$. Then $\mathbf{E}[\mathcal{C}(\mathbf{p})] \leq \mathbf{E}[\mathcal{C}(\mathbf{q})]$.*

For any integer k , define the k -th harmonic number by $\mathcal{H}_k := \sum_{i=1}^k (1/i)$. We will use the following well-known and elementary bounds for $\mathcal{H}_k$.

Lemma 2.2. *For any $k \in \mathbb{N}$, $\ln k \leq \mathcal{H}_k \leq \ln k + 1$. Moreover, for any two positive integers k_1, k_2 we have $\mathcal{H}_{k_1+k_2} \leq \mathcal{H}_{k_1} + \mathcal{H}_{k_2}$.*

2.2 Randomized Broadcast

Let $T = (V, E)$ a connected tree with n vertices. Let s be the root of this tree. For a vertex $v \in V$, $\deg(v)$ is the degree of v and $\text{height}(T)$ is the height of T . For any $0 \leq i \leq \text{height}(T) - 1$, let L_i be the i -th level of the graph, i.e., $L_i := \{u \in V \mid \text{dist}(u, s) = i\}$.

¹ The cover time is the expected time required by a random walk to visit all vertices of a given graph.

Our randomized broadcast process works as follows. Assume that initially only the root $s \in V$ is informed, i. e. , it stores the message r . In every step $1, 2, \dots$, each informed node chooses a neighbor uniformly at random and sends a copy of r to it. Denote by $I(t)$ the set of informed vertices at the end of step t . Hence, $I(0) = \{s\}$.

We also define a *levelled broadcast algorithm*, which can be regarded as a slowed-down version of the randomized broadcast protocol described above. The algorithm works in $\text{height}(T)$ phases. Phase i ($1 \leq i \leq \text{height}(T)$) is responsible to inform all nodes of level L_i of T . In Phase 1 only s is active and sends r to a random neighbour. The phase ends as soon as all neighbours of s are informed. Generally, we assume that in Phase i only the nodes of level $i - 1$ are active, meaning in this phase only these nodes send r to an i.u.r. chosen neighbour. Phase i ends as soon as *all* vertices of L_i are informed. Phase $i + 1$ starts immediately in the next step. The random variable Δ_i counts the number of steps of Phase i . Note that at each time step t of Phase i , $L_j \subseteq I_t$ for each $j \leq i$, but $L_j \cap I_t = \emptyset$ for each $j > i$.

Let $\text{RB}_s(T) = \min\{t \in \mathbb{N} : I(t) = V \mid I(0) = s\}$ be a random variable counting the number of steps it takes the random broadcast algorithm to inform all vertices if the broadcast is initialized by node s . Similarly, let $\text{LB}_s(T) = \min\{t \in \mathbb{N} : I_{\text{LB}}(t) = V \mid I_{\text{LB}}(0) = s\}$ be the runtime of the levelled broadcast algorithm. As before, the corresponding expected runtimes are denoted as $\mathbf{E}[\text{RB}_s(T)]$ and $\mathbf{E}[\text{LB}_s(T)]$, respectively. It is immediate from the definition that $\text{RB}_s(T)$ is stochastically dominated by $\text{LB}_s(T)$, in particular, $\mathbf{E}[\text{RB}_s(T)] \leq \mathbf{E}[\text{LB}_s(T)]$ for any tree T and vertex $s \in V$.

3 Coupon Collector's Problem

3.1 Simple Approximations

Now let us define $\mathcal{L}_1(\mathbf{p}) := \sum_{i=1}^n \frac{1}{(n-i+1) \cdot p_i}$ and $\mathcal{L}_2(\mathbf{p}) := \frac{\mathcal{H}_n}{n} \cdot \sum_{i=1}^n \frac{1}{p_i}$. First we give an example for which both bounds are rather weak.

Lemma 3.1. *Define a vector of length n by $\mathbf{p} := (\frac{1}{n^2}, \frac{1}{n-1} - \frac{1}{(n-1)n^2}, \dots, \frac{1}{n-1} - \frac{1}{(n-1)n^2})$. Then $\mathbf{E}[\mathcal{C}(\mathbf{p})] = \Theta(n^2)$, but $\mathcal{L}_1(\mathbf{p}) = \mathcal{O}(n \log n)$ and $\mathcal{L}_2(\mathbf{p}) = \mathcal{O}(n \log n)$.*

The following lemma gives a better way of approximating $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ through $\mathcal{U}_1 := \frac{\mathcal{H}_n}{p_1}$.

Lemma 3.2. *For every probability vector $\mathbf{p}$, $\frac{1}{\mathcal{H}_n} \cdot \mathcal{U}_1 \leq \mathbf{E}[\mathcal{C}(\mathbf{p})] \leq \mathcal{U}_1$. Moreover, there exists a probability vector $\mathbf{p}$ such that $\mathbf{E}[\mathcal{C}(\mathbf{p})] = \mathcal{O}\left(\frac{1}{\mathcal{H}_n} \cdot \mathcal{U}_1\right)$.*

3.2 Approximation Ratio $\Theta(\log \log n)$

In this section we study the upper bound $\mathcal{U}_2(\mathbf{p}) = \sum_{i=1}^n \frac{1}{i \cdot p_i}$. A nice intuitive upper bound is the following bound $\mathcal{U}_3(\mathbf{p}) := \sum_{i=1}^n \frac{1}{\sum_{k=1}^i p_k}$. $\mathcal{U}_3(\mathbf{p})$ that can be regarded as an upper bound of a slightly modified coupon-collecting process. The number of coupons in this process is the same, but whenever the i th new coupon is drawn we exchange the coupon with the coupon that is drawn with probability p_{n-i+1} . Hence, we assume that coupons with large values of p_i are chosen first. One can prove that $\mathcal{U}_2(\mathbf{p})$

and $\mathcal{U}_3(\mathbf{p})$ differ only by a factor of at most two. Hence, the results in this section hold for both bounds and we can use the simpler version $\mathcal{U}_2(\mathbf{p})$ for our results in this section.

To show that $\mathcal{U}_2(\mathbf{p})$ (resp. $\mathcal{U}_3(\mathbf{p})$) approximates $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ up to a factor $\mathcal{O}(\log \log n)$ we need the following elementary lemma. The lemma states that $\mathcal{U}_2(\mathbf{p})$ is mainly determined by coupons that have a small probability to be drawn.

Lemma 3.3. *If there is a $1 \leq j \leq n$ with $p_1 \leq p_j / \ln n$, then $\mathcal{U}_2(\mathbf{p}) \leq 3 \sum_{i=1}^{j-1} \frac{1}{i \cdot p_i}$.*

We can now prove an upper bound of $\mathcal{O}(\log \log n)$ on the approximation ratio.

Theorem 3.4. *For every probability vector $\mathbf{p}$, $\frac{1}{3e \cdot \ln \ln n} \cdot \mathcal{U}_2(\mathbf{p}) \leq \mathbf{E}[\mathcal{C}(\mathbf{p})] \leq 2\mathcal{U}_2(\mathbf{p})$.*

Proof. It is not difficult to see that $\mathcal{U}_3(\mathbf{p}) \leq 2\mathcal{U}_2(\mathbf{p})$ which immediately implies by using $\mathbf{E}[\mathcal{C}(\mathbf{p})] \leq \mathcal{U}_3(\mathbf{p})$ that $\mathbf{E}[\mathcal{C}(\mathbf{p})] \leq 2\mathcal{U}_2(\mathbf{p})$. To prove the remaining inequality, we divide the coupons in groups G_i with $i \in \{1, \dots, \ln \ln n\}$ by

$$G_i := \{1 \leq i \leq n \mid e^{i-1} \cdot p_1 \leq p_i < e^i \cdot p_1\}.$$

All remaining coupons j with $p_j > p_1 \cdot \ln n$ will be in G_0 . We set $g_i := |G_i|$ and

$$\Phi(\mathbf{p}) := \max_{1 \leq i \leq \ln \ln n} \left\{ \frac{1}{e^i \cdot p_1} \cdot \mathcal{H}_{g_i} \right\}.$$

Claim. For every probability distribution, $\Phi(\mathbf{p}) \leq \mathbf{E}[\mathcal{C}(\mathbf{p})]$.

Proof. Fix i with $1 \leq i \leq \ln \ln n$ and consider the coupon collecting problem restricted to group G_i . Obviously $\mathbf{E}[\mathcal{C}(\mathbf{p})] \geq \mathbf{E}[\mathcal{C}(G_i)]$ (where $\mathcal{C}(G_i)$ is the time to get all coupons from group G_i) and

$$\mathbf{E}[\mathcal{C}(G_i)] \geq \sum_{k=1}^{g_i} \frac{1}{\sum_{j=1}^{g_i-k} p_{j+\sum_{l=1}^{k-1} g_l}} \geq \sum_{k=1}^{g_i} \frac{1}{\sum_{j=1}^k e^i \cdot p_1} \geq \frac{1}{e^i \cdot p_1} \cdot \mathcal{H}_{g_i}. \quad \square$$

Claim. For every probability distribution, $\mathcal{U}_2(\mathbf{p}) \leq 3e \ln \ln n \cdot \Phi(\mathbf{p})$.

Proof. By Lemma 3.3 and Lemma 2.2 we get

$$\begin{aligned} \mathcal{U}_2(\mathbf{p}) &\leq 3 \sum_{k=1}^{\ln \ln n} \sum_{j=1}^{g_i} \frac{1}{p_{j+\sum_{l=1}^{k-1} g_l}} \cdot \frac{1}{j + \sum_{l=1}^{k-1} g_l} \\ &\leq 3 \sum_{k=1}^{\ln \ln n} \sum_{j=1}^{g_i} \frac{1}{e^{k-1} \cdot p_1} \cdot \frac{1}{j + \sum_{l=1}^{k-1} g_l} \\ &= 3 \sum_{k=1}^{\ln \ln n} \frac{1}{e^{k-1} \cdot p_1} \cdot \left(\mathcal{H}_{\sum_{j=1}^k g_j} - \mathcal{H}_{\sum_{j=1}^{k-1} g_j} \right) \\ &\leq 3 \sum_{k=1}^{\ln \ln n} \frac{1}{e^{k-1} p_1} \cdot \mathcal{H}_{g_k} \leq 3 \sum_{k=1}^{\ln \ln n} e \cdot \Phi(\mathbf{p}). \quad \square \end{aligned}$$

Combining both claims proves the theorem. $\square$

The next result shows that the bound from Theorem 3.4 is tight.

Theorem 3.5. *There are probability vectors $\mathbf{p}$ with $\mathcal{U}_2(\mathbf{p}) = \Omega(\ln \ln n) \cdot \mathbf{E}[\mathcal{C}(\mathbf{p})]$.*

3.3 Approximation Ratio $\mathcal{O}(\ln \ln \ln n)$

First order the terms of $\Phi(\mathbf{p})$ non-increasingly. More precisely, let $j_1, j_2, \dots, j_{\ln \ln n}$ be the order of the terms such that $\frac{1}{e^{j_i p_1}} \mathcal{H}_{g_{j_i}} \geq \frac{1}{e^{j_{i+1} p_1}} \mathcal{H}_{g_{j_{i+1}}}$. Define

$$\mathcal{U}_4 := \sum_{i=1}^{\ln \ln n} \frac{1}{i} \frac{1}{e^{j_i-1} p_1} \mathcal{H}_{g_{j_i}}.$$

Theorem 3.6. *For every probability vector $\mathbf{p}$, $\frac{1}{\ln \ln \ln n} \mathcal{U}_4(\mathbf{p}) \leq \mathbf{E}[\mathcal{C}(\mathbf{p})] \leq 35 \mathcal{U}_4(\mathbf{p})$.*

Proof. First observe that $\mathcal{U}_4(\mathbf{p}) \leq (\ln \ln \ln n) \cdot \Phi(\mathbf{p})$. Since $\Phi(\mathbf{p}) \leq \mathbf{E}[\mathcal{C}(\mathbf{p})]$, it only remains to prove that $\mathbf{E}[\mathcal{C}(\mathbf{p})] \leq 35 \mathcal{U}_4(\mathbf{p})$. Define X as the random variable counting the number of missing coupons from the groups G_{j_i} with $2 \leq i \leq \ln \ln n$ after $5e \cdot \mathcal{U}_4(\mathbf{p})$ many steps. Then

$$\begin{aligned} \mathbf{E}[X] &\leq \sum_{i=2}^{\ln \ln n} g_{j_i} \cdot (1 - e^{j_i-1} p_1)^{5e \sum_{k=1}^{\ln \ln n} (1/k) \cdot 1/(e^{j_k} p_1) \cdot \mathcal{H}_{g_{j_k}}} \\ &\leq \sum_{i=2}^{\ln \ln n} g_{j_i} \cdot (1 - e^{j_i-1} p_1)^{5 \ln i / (e^{j_i-1} p_1) \cdot \mathcal{H}_{g_{j_i}}} \\ &\leq \sum_{i=2}^{\ln \ln n} g_{j_i} \cdot e^{-5 \ln i \cdot \mathcal{H}_{g_{j_i}}} = \sum_{i=2}^{\ln \ln n} \frac{g_{j_i}}{i^{5 \mathcal{H}_{g_{j_i}}}} \leq \sum_{i=2}^{\ln \ln n} \frac{1}{i^4} < \frac{1}{2}. \end{aligned}$$

Using Markov's inequality we can show that $\Pr[X \geq 2 \mathbf{E}[X]] \leq \frac{1}{2}$, and therefore $\Pr[X = 0] \geq \frac{1}{2}$. Hence the expected time until all coupons from the groups G_{j_i} ($2 \leq i \leq \ln \ln n$) are drawn is bounded by $2 \cdot 5e \mathcal{U}_4(\mathbf{p})$. The expected time to collect all coupons in G_{j_1} is clearly bounded by $\mathcal{U}_4(\mathbf{p})$. Consider now a coupon j that is not in one of the G_i with $1 \leq i \leq \ln \ln n$. In this case $p_j \geq p_1 \cdot \ln n$. Since $\mathcal{U}_4(\mathbf{p}) \geq \frac{1}{p_1}$ we can bound the probability that this coupon has not been collected after $4 \cdot \mathcal{U}_4(\mathbf{p})$ steps as follows:

$$(1 - p_j)^{\frac{4}{p_1}} \leq (1 - p_1 \ln n)^{\frac{4}{p_1}} \leq e^{-4 \ln n} \leq n^{-4}.$$

Putting everything together, we can conclude that the expected time to get all n coupons is bounded above by $10e \cdot \mathcal{U}_4(\mathbf{p}) + \mathcal{U}_4(\mathbf{p}) + 5 \cdot \mathcal{U}_4(\mathbf{p}) \leq 35 \cdot \mathcal{U}_4(\mathbf{p})$. $\square$

4 Randomized Broadcasting

In this section we consider the simple randomized broadcast algorithm on trees.

We show that the expected broadcast time for the star is not smaller than that of any tree with the same number of nodes. For the proof, we first derive inequalities relating certain sums of weighted coupon collector's times. Roughly speaking, we use them to show that for any given tree (different from the star), there is a sequence of transformations into a star graph such that each transformation does only increase the expected broadcast time.

We have to introduce a little bit further notation. Let $\mathbf{p}$ be the probability vector of length n with $\mathbf{p} = (1/n, \dots, 1/n)$. Then we define $\mathcal{C}_n := \mathcal{C}(\mathbf{p})$ as the standard (uniform) coupon collecting time. A slightly different variant arises when we are only interested in obtaining the first $n - 1$ coupons. To this end, let $\mathbf{q}$ be vector of length $n - 1$ with $(1/n, \dots, 1/n)$ and define $\mathcal{C}_n^{-1} := \mathcal{C}(\mathbf{q})$. We observe that $\mathbf{E}[\mathcal{C}_n^{-1}] = \frac{n}{n-1} \mathbf{E}[\mathcal{C}_{n-1}] = n \cdot \mathcal{H}_{n-1}$. By elementary calculations involving harmonic numbers, we prove:

Lemma 4.1. *Let $a, b \geq 2$ be two arbitrary integers. Then $\mathbf{E}[\mathcal{C}_a^{-1}] + \mathbf{E}[\mathcal{C}_b^{-1}] < \mathbf{E}[\mathcal{C}_{a+b}^{-1}]$ and $\mathbf{E}[\mathcal{C}_a] + \mathbf{E}[\mathcal{C}_b^{-1}] < \mathbf{E}[\mathcal{C}_{a+b-1}]$.*

The first statement of Lemma 4.1 immediately implies:

Corollary 4.2. *Let $a_1, a_2, \dots, a_n$ be arbitrary integers ≥ 2 and let $a = a_1 + a_2 + \dots + a_n$. Then, $\mathbf{E}[\max_{k=1}^n \mathcal{C}_{a_k}^{-1}] < \mathbf{E}[\mathcal{C}_a^{-1}]$.*

We now prove two lemmas which are illustrated in Figure 2 and 3.

Lemma 4.3. *Let T_1 be a tree with n vertices. For $i \geq 2$ let $L_i = \{v_1, v_2, \dots, v_{|L_i|}\}$ be the i -th level of T_1 . Let T_2 be the tree obtained from T_1 by rooting all the successors of $\{v_2, v_3, \dots, v_{|L_i|}\}$ to v_1 . Then, $\mathbf{E}[\Delta_i(T_1)] < \mathbf{E}[\Delta_i(T_2)]$.*

Proof. By definition of the levelled broadcast algorithm and linearity of expectations, it is sufficient to consider phase i and argue that T_2 requires more time than T_1 (in expectation). Let $d = \deg(v_1) + \deg(v_2) + \dots + \deg(v_{|L_i|})$. By Corollary 4.2,

$$\mathbf{E}[\Delta_i(T_1)] = \mathbf{E}\left[\max_{k=1}^{|L_i|} \mathcal{C}_{\deg(v_k)}^{-1}\right] < \mathbf{E}[\mathcal{C}_d^{-1}] = \mathbf{E}[\Delta_i(T_2)]. \quad \square$$

With similar techniques we prove:

Lemma 4.4. *Let T_1 be a tree with n vertices such that $h = \text{height}(T_1) \geq 3$. Assume that in level i , $1 \leq i \leq \text{height}(T_1) - 1$, there is a unique node w_i on level i with a successor. Let T_2 be the tree obtained from T_1 by rooting all successors of w_{h-1} to w_{h-2} . Then*

$$\mathbf{E}[\Delta_{h-1}(T_1)] + \mathbf{E}[\Delta_h(T_1)] < \mathbf{E}[\Delta_{h-1}(T_2)].$$

Now we are ready to show the main result of this section.

Theorem 4.5. *The star strictly maximizes the expected broadcast time among all trees, i.e., any other tree has a strictly smaller expected broadcast time.*

Proof. To prove Theorem 4.5, we show that for any given tree $T = T_1$, there is a sequence of simple alterations resulting in a sequence of trees $T_1, \dots, T_k$ such that T_k is the star graph and for any i with $1 \leq i \leq k$

$$\mathbf{E}[\text{LB}_s(T_i)] \leq \mathbf{E}[\text{LB}_s(T_{i+1})],$$

and one of these inequalities is strict. Since we have $\mathbf{E}[\text{RB}_s(T_1)] \leq \mathbf{E}[\text{LB}_s(T_1)]$ and $\mathbf{E}[\text{LB}_s(T_k)] = \mathbf{E}[\text{RB}_s(T_k)]$, this implies the theorem.

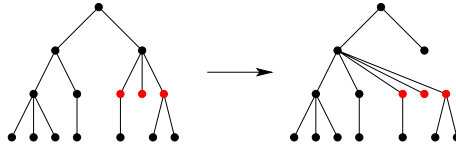


Fig. 2. Illustration of the operation considered in Lemma 4.3; the corresponding inequality is $\mathbf{E}[C_3^{-1}] + \mathbf{E}[C_4^{-1}] < \mathbf{E}[C_6^{-1}]$

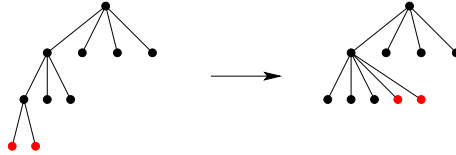


Fig. 3. Illustration of the operation considered in Lemma 4.4. Also here, the corresponding inequality is $\mathbf{E}[C_4^{-1}] + \mathbf{E}[C_3^{-1}] < \mathbf{E}[C_6^{-1}]$.

Let $h = \text{height}(T) \geq 3$. We start with $T_1 = T$ and apply Lemma 4.3 on every level of T_1 , considering the levels in increasing order. This results in $T_2, \dots, T_h$. For T_i ($1 \leq i \leq h$) there is at most one vertex with a successor on levels $0, \dots, i-1$. Moreover, we have

$$\mathbf{E}[\text{LB}_s(T_1)] \leq \mathbf{E}[\text{LB}_s(T_2)] \leq \dots \leq \mathbf{E}[\text{LB}_s(T_h)].$$

Now we apply Lemma 4.4 on all levels of T_d , considering all levels in decreasing order. This results in $T_{h+1}, \dots, T_{2h-1}$. For T_{h+i} ($1 \leq i \leq h-1$), T_{h+i} is a tree with $\text{height}(T_{h+i}) = h-i$. Hence, T_{2h-1} is the star. We have

$$\mathbf{E}[\text{LB}_s(T_{h+1})] \leq \mathbf{E}[\text{LB}_s(T_{h+2})] \leq \dots \leq \mathbf{E}[\text{LB}_s(T_{2h-1})].$$

Now we apply Lemma 4.4 on all levels of T_d , considering all levels in decreasing order. This results in $T_{h+1}, \dots, T_{2h-1}$. For T_{h+i} ($1 \leq i \leq h-1$), T_{h+i} is a tree with $\text{height}(T_{h+i}) = h-i$. Hence, T_{2h-1} is the star. Again we have

$$\mathbf{E}[\text{LB}_s(T_{h+1})] \leq \mathbf{E}[\text{LB}_s(T_{h+2})] \leq \dots \leq \mathbf{E}[\text{LB}_s(T_{2h-1})],$$

which shows that the given tree $T = T_1$ has a smaller broadcast time than the star.

Let us now prove that any tree T different from the star has a *strictly* smaller broadcast time. If $h \geq 4$, the claim follows directly from Lemma 4.4 and if $h = 2$ T is a star. Hence it suffices to consider the case $h = 3$. Now if the root s has degree at least 2, Lemma 4.4 proves that the inequality is strict. But if the root s has degree 1, the tree is already the star. This completes the proof. $\square$

5 Conclusion

In this paper we investigate the weighted coupon collector's problem. In the first part we suggest two upper bounds for the expected time to draw a copy of every coupon and we

rigorously analyze the approximation ratios of the suggested bounds. Despite their simplicity, we show that our first bound achieves an approximation ratio of $\Theta(\log \log n)$. and an extension of it improves the approximation ratio to $\mathcal{O}(\log \log \log n)$.

In the second part we derive bounds on $\mathbf{E}[\mathcal{C}(\mathbf{p})]$ for slight variations of the weighted coupon collector's problem. We apply these bounds inductively to prove that the star graph is the only tree which maximizes the broadcast time. For the proof of this result the assumption that the network is a tree is crucial, but it is plausible to conjecture that the star is also the worst-case graph among *all* graphs.

References

1. Adler, I., Oren, S., Ross, S.: The coupon collector's problem revisited. *Journal of Applied Probability* 40(2), 513–518 (2003)
2. Brightwell, G., Winkler, P.: Extremal cover times for random walks on trees. *Journal of Graph Theory* 14(5), 547–554 (1990)
3. Broder, A., Karlin, A.: Bounds on the cover time. In: 29th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1988), pp. 479–487 (1988)
4. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: 6th Annual ACM-SIGOPT Principles of Distributed Computing (PODC 1987), pp. 1–12 (1987)
5. Elsässer, R., Sauerwald, T.: On the runtime and robustness of randomized broadcasting. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 349–358. Springer, Heidelberg (2006)
6. Feige, U.: A Tight Lower Bound for the Cover Time of Random Walks on Graphs. *Random Structures and Algorithms* 6(4), 433–438 (1995)
7. Feige, U.: A Tight Upper Bound for the Cover Time of Random Walks on Graphs. *Random Structures and Algorithms* 6(1), 51–54 (1995)
8. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized Broadcast in Networks. *Random Structures and Algorithms* 1(4), 447–460 (1990)
9. Flajolet, P., Gardy, D., Thimonier, L.: Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics* 39, 207–229 (1992)
10. Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized Rumor Spreading. In: 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2000), pp. 565–574 (2000)
11. Ross, S.: *Stochastic Processes*. Wiley & Sons, Chichester (1996)
12. Ross, S.: *Introduction to Probability Models*, 8th edn. Academic Press, London (2003)
13. Stirzaker, D.: *Elementary Probability*, 2nd edn. Cambridge University Press, Cambridge (2003)

Sublinear-Time Algorithms for Tournament Graphs

Stefan Dantchev¹, Tom Friedetzky^{1,*}, and Lars Nagel¹

Department of Computer Science, Durham University, Durham DH1 3LE, U.K.

Abstract. We show that a random walk on a tournament on n vertices finds either a sink or a 3-cycle in expected time $O(\sqrt{n} \cdot \log n \cdot \sqrt{\log^* n})$, that is, sublinear both in the size of the description of the graph as well as in the number of vertices. This result is motivated by the search of a generic algorithm for solving a large class of search problems called Local Search, *LS*. *LS* is defined by us as a generalisation of the well-known class *PLS*.

Keywords: sublinear-time algorithms, tournament, random walk.

1 Introduction

A tournament is a complete oriented graph. It is called transitive if it contains no directed cycles. A sink is a vertex with out-degree 0. (For extensive definitions see def. 1 and 2.) Assume that a computational model is given so that it is possible to query the out-degree of a vertex and to sample a random outgoing edge in constant time. We study the following question: Given a tournament on n vertices can we find either a sink or a 3-cycle in time sublinear in n ? This question is closely related to a generalisation of the well-known class of polynomial search problems, which we explain below. We answer the question positively in the given computational model, which essentially allows for a random walk on the vertices of the tournament graph.

1.1 Motivation

A *Local Search Problem* LS_t is defined on an exponentially big universe $S_t = \{0, 1\}^t$. There is a neighbourhood relation $N \subseteq S_t \times S_t$, and a cost function $c : S_t \rightarrow \mathbb{N}$. An algorithm that solves the LS_t should, at the very least, be able to

1. calculate the cost function $c(x)$ for a given $x \in S_t$;
2. make a local move, i.e. for any $x \in S_t$ either find $y \in S_t$ such that $y \neq x$, $N(x, y)$, and $c(y) \leq c(x)$ or realise that no such y exists. In the latter case, x is a point of a local minimum, and the algorithm terminates.

* Partially supported by EPSRC grant “Property Testing”.

Note that the search algorithm may not terminate if there is a finite sequence of points $x_1, x_2, \dots, x_k$ such that $N(x_i, x_{i+1})$ for every $i = 1 \dots k-1$, $N(x_k, x_1)$, and $c(x_1) = c(x_2) = \dots = c(x_k)$. However, if a strict inequality $c(y) < c(x)$ is required in the definition of a local move above then the algorithm always terminates in a local minimum after at most $n = 2^t$ steps.

It is not hard to see that our definition includes the well-known Polynomial Search Problems, *PLS*, introduced in [3]. We shall explain this on an example.

The HappyNet problem is defined as follows. An undirected graph $G = (V, E)$ on n vertices is given together with a measure $w : E \rightarrow \mathbb{Z}$ of how any two adjacent vertices like one another. For any bisection $b : V \rightarrow \{-1, +1\}$ of the set of vertices, the happiness of a vertex u is defined to be $\sum_{\{u,v\} \in E} b_u b_v w_{\{u,v\}}$. The question is if there exists a bisection such that every vertex is happy, i. e. has a non-negative happiness.

The answer is yes (always), and a happy bisection can be found by a simple greedy local search:

1. Start with any bisection (say with everyone being in group $+1$).
2. As long as there is an unhappy vertex, move it to the other group.

It is easy to see why the HappyNet is a Local Search Problem (in fact even a PLS): The universe is $S_t = \{-1, +1\}^t$, the neighbourhood predicate $N(b_1, b_2)$ contains precisely the pairs of bisections b_1, b_2 , which are 1 apart in Hamming distance, and the cost function $c(b)$ is the total happiness $\sum_{\{u,v\} \in E} b_u b_v w_{\{u,v\}}$ (this is the crucial observation in proving that the greedy local search works).

Let us now consider the HappyNet problem on directed graphs. A very simple example, the directed 3-cycle with equal costs of dislike (say -1 on all three directed edges), shows that a happy configuration does not necessarily exist. Moreover, a local search algorithm does not terminate as it goes into some cycle of total happiness -1 .

We shall finally give the reduction of any Local Search Problem to the question of

Finding a sink or a 3-cycle in a tournament. Indeed, given an LS_t , we can define a tournament on $n = 2^t$ vertices as follows.

$$\begin{aligned} \forall x, y \quad (y \prec x) &\rightarrow (x \neq y \wedge N(x, y) \wedge c(y) \leq c(x)) \\ \forall x, y, z \quad (x \prec y) \wedge (y \prec z) &\rightarrow (x \prec z) \\ \forall x \exists y \quad y \prec x \end{aligned}$$

This set of formulae is inconsistent over a finite universe as it claims that there is a total order with no minimal element. The first line constitutes the actual reduction. A violation of the second one implies the existence of a 3-cycle while a violation of the third line implies the existence of a global minimum in the tournament (which is a local minimum in the Local Search Problem).

1.2 Related Work and Our Result

It is a straightforward observation (or a folklore result) that in a *transitive tournament* on n vertices, the (unique) sink can be found in time $O(\sqrt{n})$ by a randomised algorithm with a constant probability of success. Indeed, one can first sample $\sqrt{n}$ vertices uniformly at random, and then walk $\sqrt{n}$ steps from the smallest one (which can be found in $\sqrt{n} - 1$ comparisons). The probability that all sampled vertices are at least $\sqrt{n}$ away from the sink is $\left(\frac{n-\sqrt{n}}{n}\right)^{\sqrt{n}} \approx \frac{1}{e}$, and thus this algorithm succeeds in time $O(\sqrt{n})$ with probability $1 - \frac{1}{e}$. By repeating it until the sink is found, one gets an algorithm that runs in expected $O(\sqrt{n})$ time (and always succeeds).

In the paper, we consider the more general case of an arbitrary, i. e. *not necessarily transitive*, tournament. In this case, we show that a random walk on the tournament graph finds either a sink or a 3-cycle in expected time $O(\sqrt{n} \log n \sqrt{\log^* n})$. This gives a generic algorithm for solving an instance of LS_t in time $O^*(2^{1/2})$, which is better than a trivial enumeration. Unfortunately, the random walk can be carried out only if we assume that we can make not only a local move but also sample uniformly at random the set of all possible local moves. It is an open question if an algorithm exists that can work by making local moves and beat the trivial enumeration.

Tournaments and sublinear algorithms. Good introductions into the topic of tournament graphs and compilations of the results until 1978 are given by Moon [4] and Reid and Beineke [5]. We use a few results from Moon's book [4]. More work on tournaments was published in the years after, but as it seems nothing close to our result.

Surveys of sublinear-time algorithms are given by Rubinfeld [6] and Czumaj and Sohler [2]. Algorithms that run in sublinear time cannot read the whole input. Usually some combinatorial object is given, and the algorithm can query the object in random locations. Two types of algorithms are distinguished: Las Vegas algorithms run in (normally expected) sublinear time and always return the correct result. Monte Carlo algorithms can err with a certain probability, but are usually faster.

We will introduce a Monte Carlo algorithm that finds indication of a sink or cycle in time $O(\log(n))$, but does not return a witness. Our main result includes a Las Vegas algorithm that returns a witness for a sink or cycle in expected time $O^*(\sqrt{n})$.

1.3 Outline

In section 2 we will define tournament graphs, prove useful properties they have and introduce notation used in the subsequent sections. Section 3 describes the algorithm SIMPLESINK that runs in $O(\log(n))$, but finds only indication of a sink or cycle and does not return a witness. Section 4 is dedicated to the main result and presents an algorithm EXTENDED SIMPLESINK that returns a witness for a sink or a cycle in time $O(\sqrt{n} \cdot \log n \cdot \sqrt{\log^* n})$.

2 Tournaments

Definition 1. A tournament is a complete oriented graph. Thus, a tournament is a directed graph so that for each pair of vertices u, v , either (u, v) or (v, u) is an edge.

Such a graph is called a tournament because it represents an outcome of a round-robin tournament without draws. In a round-robin tournament each participant plays every other participant, and each game has a winner. If the participants are represented as vertices and the games as directed edges, the result is a tournament graph. An edge (u, v) implies that u won against v .

Definition 2. The score vector of a tournament is the ordered n -vector of its vertices' out-degrees (or scores). Throughout this paper we use both $d(v)$ and $d_{out}(v)$ to denote the out-degree of vertex v and $d_{in}(v)$ to denote its in-degree. We call a vertex v a sink if $d_{out}(v) = 0$.

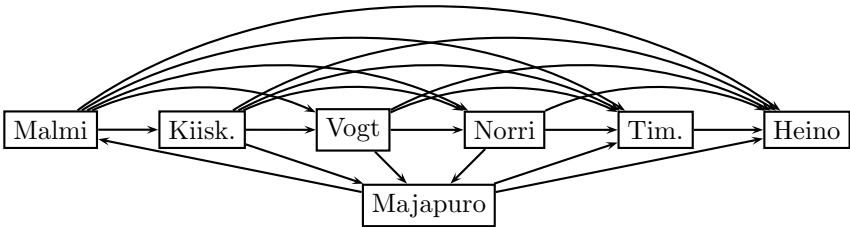
A tournament $G = (V, E)$ is transitive if for all $u, v, w \in G$, $(u, v) \in E$ and $(v, w) \in E$ imply $(u, w) \in E$.

Example 1. Championships in many sports are held as round-robin tournaments. One example that does not allow draws is curling. We show the first round's table of the "Finnish Women Curling Championship League 2007-2008":

Table 1. Example for round-robin tournament: Finnish Curling League

Teams	Malmi	Kiisk.	Vogt	Norri	Tim.	Maj.	Heino
Malmi		12-5	12-4	9-8	10-3	6-10	11-2
Kiiskinen	5-12		8-3	12-1	11-4	13-8	14-0
Vogt	4-12	3-8		11-6	7-5	11-2	13-4
Norri	8-9	1-12	6-11		15-5	7-6	13-4
Timonen	3-10	4-11	5-7	5-15		3-10	12-4
Majapuro	10-6	8-13	2-11	6-7	10-3		11-1
Heino	2-11	0-14	4-13	4-13	4-12	1-11	

Vertex "Heino" is a sink, "Malmi", "Vogt" and "Majapuro" form a 3-cycle. The subtournament that results from removing the vertex "Majapuro" is a transitive tournament.



We will use some simple observations which are well-known ([1], [4]):

Observation 1

1. In a tournament, $d_{in}(v) + d_{out}(v) = n - 1$.
2. The average in-degree / out-degree is $\frac{n-1}{2}$.
3. A tournament has at most one sink.
4. Every induced subgraph of a tournament is a tournament itself.
5. If a tournament contains any cycle then it must also contain a 3-cycle.
6. In a tournament graph there are at most $2 \cdot d + 1$ vertices with out-degree at least $n - 1 - d$.
7. For a tournament $G = (V, E)$ the following statements are equivalent:
 - (a) G is transitive.
 - (b) G has no cycles.
 - (c) The scores (out-degrees) are pairwise distinct. Thus, G 's score vector is $\{0, 1, \dots, n - 1\}$.
 - (d) For all edges $(u, v) \in E$ the following is true: $d_{out}(u) > d_{out}(v)$.
8. Every tournament has a sink or is cyclic, maybe both.

Proof. (1)–(4) are trivial (for (3): there cannot be two or more sinks as any two vertices are connected by an edge).

(5) Any chord in any cycle will, together with precisely one of cycle's segments between the chord's endpoints, define a smaller cycle. By repetition we will eventually obtain a 3-cycle.

(6) Suppose there is a subset D of $2d+2$ vertices with degree at least $n - 1 - d$. Then there are at least $(2d+2) \cdot (n - 1 - d) - \frac{(2d+2) \cdot (2d+1)}{2} = (2d+2) \cdot (n - 2d - \frac{3}{2})$ edges leaving D . That is a contradiction to the fact that there are only $|D| \cdot |V \setminus D| = (2d+2) \cdot (n - 2d - 2)$ many edges between D and $V \setminus D$. On the other hand, if D , consisting of vertices of degree at least $n - 1 - d$, has size $2d+1$, the number of edges leaving D is $(2d+1) \cdot (n - 1 - d) - \frac{(2d+1) \cdot (2d+1)}{2} = (2d+1) \cdot (n - 2d - 1) = |D| \cdot |V \setminus D|$.

(7) (a) $\rightarrow$ (b) Due to the transitivity condition G contains no triangles, and due to (5) it therefore cannot contain cycles. (b) $\rightarrow$ (c) Suppose there are two vertices u and v that have the same out-degree. There must be an edge between them. Let us say it goes from u to v , then there must be at least one vertex w that receives an edge from v and sends one to u . $u \rightarrow v \rightarrow w \rightarrow u$ is obviously a 3-cycle.

(8) follows from the previous. $\square$

Definition 3 (Notation). In the following we will write d_{max}^Z , d_{min}^Z and d_{avg}^Z respectively for the maximal, minimal and average out-degree in a set Z of vertices. Given a graph $G = (V, E)$ the number of (positive) neighbours of a vertex $z \in V$ in a subset $Q \subset V$ will be denoted $d(z, Q) = |\{q \in Q \mid (z, q) \in E\}|$. Accordingly, $d(Z, Q) = |\{q \in Q \mid \exists z \in Z : (z, q) \in E\}|$ is the size of the neighborhood of set Z in Q . Finally, $E(Z, Q)$ denotes the set of edges that start in Z and end in Q .

The computational model we use allows for picking random outgoing edges. Access to out-degree queries would be nice to possibly reduce the running time of SIMPLESINK (section 3), but they are not necessary.

3 Algorithm That Returns No Witness

In this section we introduce the Monte Carlo algorithm SIMPLESINK (see figure 1) that decides for a given graph if it contains a sink or a cycle, but does not provide a witness. The algorithm's running time is upper bounded by $O(\log(n))$, and it is allowed to err with small probability.

```

Start the random walk in an arbitrary vertex  $u$ 
Repeat  $c \cdot \log(n)$  times
  If  $d_{\text{out}}(u) = 0$  then
    Return "sink"
  Choose a random outgoing edge  $(u, v)$ 
   $u \leftarrow v$ 
Return "cycle"

```

Fig. 1. Algorithm SIMPLESINK

The random walker in SIMPLESINK makes at most $c \log(n)$ steps (where c is constant) and halts earlier only if it finds a sink. In the former case it returns "cycle", in the latter case it returns "sink". To possibly reduce the running time, the out-degree could be queried in each step; if the out-degree does not decrease the algorithm could abort and return "cycle" (see Observation 1(7)). In many cases this would result in a shorter running time, but it would of course not improve the worst case.

Theorem 1. *The algorithm SIMPLESINK returns a correct result with high probability in time $O(\log(n))$.*

Proof. The algorithm can err only if the tournament contains no cycles and if the sink is not reached after $c \log n$ steps. We will prove for the cycle-free graph that the expected time to run into the sink is $O(\log n)$. Recall that for cycle-free tournaments, there must be precisely one sink. Moreover, all degrees are pairwise distinct and $d_{\text{out}}(u) > d_{\text{out}}(v)$ for every edge (u, v) .

The proof is a simple application of the stick-breaking experiment — in which one holds one end of a stick with n many notches and repeatedly breaks (what is currently left of) the stick at a randomly chosen notch and discards what is not being held; it is well known that the time until one ends up with a fragment of size 1 is $O(\log n)$ with high probability. We only need to make the constant c in the description of the algorithm large enough. $\square$

Observation 2. *Suppose it is possible to only pick a random edge but not to pick a random outgoing edge. If the tournament contains no cycles, the (thus modified) algorithm SIMPLESINK will reach the sink after no more than n steps on the average. This time can still be considered sublinear in the size of the input (number of edges), although clearly not any more in the number of vertices.*

Proof. Let us define $J(i)$ as the expected number of jumps necessary, if the random selection of the start vertex (for the random walker) is restricted to the i vertices of lowest out-degree which form a transitive subtournament. Ignoring the first jump (to the start vertex), we have $J(1) = 0 \in O(1)$.

Now consider the step from $J(i-1)$ to $J(i)$. We get the transitive subtournament of i vertices by adding the vertex of lowest out-degree from the remaining $n-i+1$ vertices to the $i-1$ vertices we had before. With probability $\frac{i-1}{i}$ the first selected vertex is one of the former vertices so that we can use the old result. With probability $\frac{1}{i}$ we will hit the new top vertex, and after we have found an outgoing edge, we have the same situation as in the former case $i-1$. The probability to hit an outgoing edge is $\frac{i-1}{n-1}$, the expected number of attempts, until we hit an outgoing edge, is $\frac{n-1}{i-1}$. Thus we get:

$$J(i) = \frac{i-1}{i} \cdot J(i-1) + \frac{1}{i} \cdot \left(\frac{n-1}{i-1} + J(i-1) \right) = J(i-1) + \frac{n-1}{i \cdot (i-1)} = (n-1) \cdot \sum_{j=2}^i \frac{1}{j \cdot (j-1)}$$

$$J(n) = (n-1) \cdot \sum_{j=2}^n \frac{1}{j \cdot (j-1)} = (n-1) \cdot \sum_{j=2}^n \frac{1}{j-1} - \frac{1}{j} = (n-1) \cdot \left(1 - \frac{1}{n} \right) < n-1 = O(n) \quad \square$$

4 Algorithm That Returns a Witness

In this section we introduce algorithm EXTENDED SIMPLE SINK (see figure 2) that finds a witness for a sink or a cycle in expected time $O(\sqrt{n} \cdot \log(n) \cdot \sqrt{\log^*(n)})$.

```

Choose random vertex  $u$  and start random walk
Repeat
  If  $d_{\text{out}}(u) = 0$  then
    Return  $(u, \text{sink})$ 
  Store  $u$ 
  If  $u$  was visited before then
     $C \leftarrow$  set of vertices between two  $u$ -visits
    Return  $(C, \text{cycle})$ 
  Choose a random outgoing edge  $(u, v)$ 
   $u \leftarrow v$ 
    
```

Fig. 2. Algorithm EXTENDED SIMPLE SINK

Obviously the worst case running time of EXTENDED SIMPLE SINK is $O(n)$ because of at most $n+1$ jumps. If the witness is to be a 3-cycle, the technique (bisection method) used in the proof hint of Observation 1(5) can be applied.

We proceed by presenting two technical lemmas that will be useful in the proof of the main theorem 2.

Lemma 1. *Consider a set B of bins, $b = |B| \geq 2$. Suppose prior to each step we cover any $\lfloor \frac{b}{c} \rfloor$ bins ($c > 1$ is a constant), subject to choosing among those*

that have been covered least often in previous rounds. Whenever we throw the ball it will be allocated to a randomly chosen uncovered bin. Then the expected time for hitting any bin twice is $O(\sqrt{b})$. The same holds if the bins to be covered are chosen at random.

Proof. If all bins were uncovered we could apply the result of the birthday problem. After $z = 11 \cdot \sqrt{b}$ balls the probability for hitting no bin twice would be $\Pr(\text{not twice}) < e^{-0.5 \cdot z \cdot (z-1)/b} < e^{-50} < 2^{-70}$.

Now we consider the partly covered set of bins. Due to the model restrictions, after no more than $\hat{c} = \lceil \frac{c}{c-1} \rceil$ throws each bin has been uncovered at least once. We can therefore conclude that $\hat{c}$ throws are at least as good as one throw in the uncovered case. $\hat{c} \cdot 11 \cdot \sqrt{|B|}$ throws result in $\Pr(\text{twice}) = 1 - \Pr(\text{not twice}) > 1 - 2^{-70}$.

Clearly, if the bins to be covered are chosen at random then the bound from above is still valid as the model used above (cover those that have been covered least often) can be viewed as an upper bound for the quantity in question. $\square$

Definition 4 (Quasi-uniform distribution). Consider a random walk on a directed graph $G = (V, E)$ and two sets of vertices X and M . In each step all outgoing edges of the current vertex are equiprobable to take. If the neighbourhood of each vertex $x \in X$ contains a constant fraction of M we say the jumps into M are quasi-uniformly distributed over M .

In the proof of the main theorem we will show an upper bound for the running time of the algorithm EXTENDED SIMPLESINK. For this we will use the first lemma and the fact that jumps into a certain set S of vertices are quasi-uniformly distributed over S .

Additionally we need evidence that the expected time to leave certain sets of vertices can be bounded by a constant. This evidence will be provided by the next lemma.

The proof of the lemma makes use of S being a subtournament and therefore complete and indirectly of S having a high conductance. The conductance of a subset $S \subset V$ ($|S| \leq \frac{|V|}{2}$) is defined as $\Phi(S) = \frac{|E(S, V \setminus S)|}{|E(S, S)|}$ and compares the number of edges leaving S with the number of edges remaining in S .

Lemma 2. Given a tournament $G = (V, E)$ and a partition of its vertex set V into S and T , consider a random walk starting in some vertex of S . If $d_{\min}^S > \frac{2}{3} \cdot |S|$, S will be left in expected constant time.

Proof. Suppose $d_{\min}^S > r \cdot |S|$. We will show that $r \geq \frac{2}{3}$ is sufficient. The induced subgraph $G[S]$ is a tournament itself. We define $\bar{d} = \frac{|S|}{q} - 1$, where $q > 1$ is constant, and recall Observation 1(6) which states that $G[S]$ has at most $2 \cdot \bar{d} + 1 = 2 \cdot \frac{|S|}{q} - 1 < |S| \cdot \frac{2}{q}$ vertices with out-degree at least $|S| - 1 - \bar{d} = |S| \cdot \frac{q-1}{q}$. Let us denote this set of vertices with $S_h = \{s \in S \mid d(s, S) \geq |S| \cdot \frac{q-1}{q}\}$. From this we can derive that in tournament G there is a subset $S_\ell = S \setminus S_h$ of size

$|S_\ell| \geq |S| \cdot \frac{q-2}{q}$ so that for each vertex $s_\ell \in S_\ell$ the following is true:

$$\begin{aligned} d(s_\ell, S) &< |S| \cdot \frac{q-1}{q} \leq \frac{d_{\min}^S}{r} \cdot \frac{q-1}{q} \\ d(s_\ell, T) &\geq d_{\min}^S - d(s_\ell, S) > d_{\min}^S - d_{\min}^S \cdot \frac{q-1}{r \cdot q} \end{aligned}$$

$$\Pr(s_\ell \rightarrow T) = \frac{d(s_\ell, T)}{d(s_\ell, T) + d(s_\ell, S)} \geq \frac{d_{\min}^S - d(s_\ell, S)}{d_{\min}^S - d(s_\ell, S) + d(s_\ell, S)} > \frac{d_{\min}^S - d_{\min}^S \cdot \frac{q-1}{r \cdot q}}{d_{\min}^S} = 1 - \frac{q-1}{r \cdot q}$$

Consider the case that we start in a vertex $s_h \in S_h$. Since $d(s_h, S) \geq |S| \cdot \frac{q-1}{q}$, the probability to jump into S_ℓ is

$$\Pr(s_h \rightarrow S_\ell \cup T) = \frac{d(s_h, S_\ell) + d(s_h, T)}{d(s_h, S) + d(s_h, T)} \geq \frac{d(s_h, S) - d(s_h, S_h)}{d(s_h, S)} > \frac{|S| \cdot \frac{q-1}{q} - |S_h|}{|S| \cdot \frac{q-1}{q}} > \frac{|S| \cdot \frac{q-1}{q} - |S| \cdot \frac{2}{q}}{|S| \cdot \frac{q-1}{q}} = \frac{q-3}{q-1}.$$

In order to guarantee that the expected number of steps to leave S is constant, it is sufficient to show that for all $s_h \in S_h$ and $s_\ell \in S_\ell$ respectively, $\Pr(s_h \rightarrow S_\ell \cup T)$ and $\Pr(s_\ell \rightarrow T)$ are constant. This is guaranteed for $q \geq 3$ and $r \geq \frac{q-1}{q}$. Thus, any constant $r \geq \frac{2}{3}$ will do. $\square$

In the main theorem we will prove an upper and a lower for the running time of the algorithm EXTENDED SIMPLESINK.

Theorem 2. *The expected running time of algorithm EXTENDED SIMPLESINK is $O(\sqrt{n} \cdot \log^*(n) \cdot \log(n))$. $\Omega(\sqrt{n})$ is a lower bound for the running time of algorithm EXTENDED SIMPLESINK.*

Proof. We shall first prove the upper bound.

We will only count the number of vertices visited by the random walker. This is sufficient, because the procedure for one jump runs in constant time. If we assume that the vertices are stored in a scalable hash table together with a link to the next vertex, then the costs for checking if a vertex has been visited and for updating the data structure are constant. In case a cycle is found, the cost for extracting it is in the same order as the number of visited vertices.

During the proof we will make use of the following *partition*: Given a tournament graph $G = (V, E)$ with $n = |V|$, a constant $k > 3$ and a strictly monotonically increasing function $m(n) = o(n)$, divide the set V of all vertices into three disjoint sets X , Y and M . M is defined by $|M| = m(n)$ and $d(u) \leq d(v) \forall u \in M, v \in X \cup Y$, that is, M is the set of the $m(n)$ many lowest-degree vertices. The set Y contains all vertices that receive at least $m(n) \cdot \frac{k-1}{k}$ edges from M . And finally, $X = V \setminus (M \cup Y)$ contains all vertices that receive fewer than $m(n) \cdot \frac{k-1}{k}$ edges from M .

In order to simplify the exposition we will assume the *maximal out-degree* $d_{\max}^M = d_{\max}^M(m(n))$ in M to be a strictly monotonic increasing function of $m(n)$. Thus, we consider infinite families of tournaments (distinguished only by different functions $d_{\max}^M(m(n))$). However, this is without loss of generality as each possible tournament belongs to (at least) one of the families.

The idea of the proof is as follows. We will distinguish two cases and show for each of them that either quasi-uniformly distributed jumps into a small part S (either M or $M \cup Y$) are frequent and we soon hit a vertex in S twice, or that the random walk is mainly restricted to this part. In the former case, the vertices between two hits form a cycle that can be returned as a witness. In the latter case, the analysis can be reduced to a smaller problem of the same type.

Since Y is defined so that each vertex in Y receives at least $m(n) \cdot \frac{k-1}{k}$ edges from M and since $m(n) \cdot d_{\max}^M$ is an upper bound for the number of edges that leave M , the size of Y is bounded by $|Y| \leq \frac{m(n) \cdot d_{\max}^M}{m(n) \cdot \frac{k-1}{k}} = d_{\max}^M \cdot \frac{k}{k-1}$. Due to $|Y| \cdot \frac{2}{3} < |Y| \cdot \frac{k-1}{k} \leq d_{\max}^M \leq d_{\min}^Y \leq d_{\text{avg}}^Y$, we can derive from Lemma 2 that Y will be left in expected constant time.

In the following we will distinguish two cases.

Case 1: $d_{\max}^M(m(n)) = O(m(n))$, $S = M \cup Y$

Case 2: $d_{\max}^M(m(n)) = \omega(m(n))$, $S = M$

At first we will show that a *constant fraction of jumps* into S is *quasi-uniformly distributed*. From Lemma 1 we know that then $O(\sqrt{|S|})$ jumps are sufficient.

First consider case 1 and note that $|S| = |M \cup Y| \leq m(n) + d_{\max}^M(m(n)) \cdot \frac{k}{k-1} = O(m(n))$. In each jump from X into M , (at least) a constant fraction $\frac{m(n)}{k}$ of M can be hit and, since S is not significantly larger than M , also a constant fraction of S .

Case 2 is somewhat more complicated. We again use that all jumps from X into M are quasi-uniformly distributed. It remains to show that a constant fraction of jumps into M is done from X .

For any sets of vertices A_1 and A_2 let $\Pr(A_1 \xrightarrow{a} A_2)$ denote the probability that the random walker changes from A_1 to A_2 within a steps and without spending time in any other set. $\Pr(A_1 \xrightarrow{a} A_1)$ is the probability that the random walker stays in A_1 for a steps. Moreover, we will use short forms $\Pr(A_1 \xrightarrow{a} A_2 \xrightarrow{a} \dots \xrightarrow{a} A_r) = \Pr(A_1 \xrightarrow{a} A_2) \cdot \Pr(A_2 \xrightarrow{a} A_3) \cdot \dots \cdot \Pr(A_{r-1} \xrightarrow{a} A_r)$, which express the probability that the random walker changes from A_1 to A_2 within $a_1 \leq a$ steps, that after these a_1 steps he changes from A_2 to A_3 within $a_2 \leq a$ steps, and so on.

Since Y is left after a constant number of steps and since $d_{\min}^Y \geq d_{\max}^M(m(n)) = \omega(m(n))$, Y is most likely left toward X after a constant number of steps. Starting in Y and considering a sequence of at most q jumps, where q is a large enough constant, the probability to leave Y toward X within these q jumps can be lower bounded by $\frac{3}{4}$. The probability for leaving toward M is still very small. We have:

$$\Pr(Y \rightarrow M) \leq \frac{m(n)}{k \cdot d_{\min}^Y} \leq \frac{m(n)}{k \cdot d_{\max}^M(m(n))} = o(1) \quad \Pr(Y \xrightarrow{q} M) = o(1)$$

$$\Pr(Y \xrightarrow{q} X) \geq \frac{3}{4}$$

$$\Pr(Y \xrightarrow{q} Y) \leq \frac{1}{4}$$

Assume that the random walker is in some vertex $y \in Y$. The probability that it will jump into M without being in X once, is bounded by (c is a constant):

$$\begin{aligned} \Pr(Y \xrightarrow{\infty} M) &= \Pr(Y \xrightarrow{q} M) + \Pr(Y \xrightarrow{q} Y \xrightarrow{q} M) + \Pr(Y \xrightarrow{q} Y \xrightarrow{q} Y \xrightarrow{q} M) + \dots \\ &= \Pr(Y \xrightarrow{q} M) \cdot (1 + \Pr(Y \xrightarrow{q} Y) + \Pr(Y \xrightarrow{q} Y \xrightarrow{q} Y) + \dots) \\ &\leq \Pr(Y \xrightarrow{q} M) \cdot \left(1 + \sum_{i=1}^{\infty} \left(\frac{1}{4}\right)^i\right) < \frac{c \cdot m(n)}{d_{\min}^Y} \leq \frac{c \cdot m(n)}{d_{\max}^M(m(n))} = o(1) \end{aligned}$$

Now assume that the random walker is in some vertex $x \in X$. We will show that the probability to jump directly into M is not significantly worse than jumping via Y into M :

$$\begin{aligned} \Pr(x \rightarrow M) &= \Omega\left(\frac{m(n)}{d(x)}\right) \\ \Pr(x \rightarrow Y \xrightarrow{\infty} M) &< \frac{|Y|}{d(x)} \cdot \frac{c \cdot m(n)}{d_{\min}^Y} = O\left(\frac{m(n)}{d(x)}\right) \end{aligned}$$

Note that the random walker will remain or return to X in all other cases (with probability $1 - \Pr(x \rightarrow M) - \Pr(x \rightarrow Y \xrightarrow{\infty} M)$). However, in the worst case we start in Y . But even then we get the estimate for the probabilities that we need:

$$\frac{\Pr(Y \rightarrow \dots \rightarrow X \rightarrow M)}{\Pr(Y \rightarrow \dots \rightarrow Y \rightarrow M)} = \frac{(1 - \Pr(Y \xrightarrow{\infty} M)) \cdot \Pr(X \rightarrow \dots \rightarrow X \xrightarrow{\infty} M)}{\Pr(Y \xrightarrow{\infty} M) + (1 - \Pr(Y \xrightarrow{\infty} M)) \cdot \Pr(X \rightarrow \dots \rightarrow Y \xrightarrow{\infty} M)} = \Omega(1)$$

Thus, also in case 2 sufficiently many jumps are done from X so that we now know that in either case $O(\sqrt{m(n)})$ jumps into S are expected to be enough to hit a vertex in M twice (and, thus, to provide us with a witness for a cycle). The overall probability to jump from $X \cup Y$ into M is at least $\frac{m(n)}{d_{\max}^{X \cup Y}}$, so that the expected number of steps for reaching M is at most $\frac{d_{\max}^{X \cup Y}}{m(n)}$. This leaves us with a running time of $O\left(\sqrt{m(n)} \cdot \frac{d_{\max}^{X \cup Y}}{m(n)}\right) = O(n/\sqrt{m(n)})$ in case S is left often enough.

If this is not the case, that is, if S is not left often enough, we can bound the expected number of steps outside of S by $O(n/\sqrt{m(n)})$ and concentrate on $G_1 = G[S]$, which is a tournament itself, so that we can partition it into X_1 , Y_1 and M_1 and apply the same arguments. If sufficiently many jumps are done into M_1 , the running time for the whole problem is still $O(n/\sqrt{m(n)})$. Otherwise further *subproblems* $G_i = (V_i, E_i)$ must be considered, but not more than $\log(n)$ many, because $|V_i| < \frac{|V_{i-1}|}{2}$ and $\frac{|V|}{2^{\log(n)}} = \frac{n}{n} = 1$.

We need to count all the steps that are made in the worst case. Let us define $G_0 = G$ and $Z_i = V_i \setminus S_i \forall i \in \{0, \dots, t\}$. Since $m(n) = o(n)$ we can assume $|Z_{i+1}| \ll |Z_i|$, $|S_{i+1}| \ll |S_i|$ and that the number of expected jumps j_i within Z_i (at least) bisects for Z_{i+1} . Let $G_t = (V_t, E_t)$ denote the subproblem, for which the according set S_t is left often enough. Then we get for all Z_i :

$$\sum_{i=0}^t j_i < \sum_{i=0}^t j_0 \cdot \left(\frac{1}{2}\right)^i < 2 \cdot j_0 = O\left(\frac{n}{\sqrt{m(n)}}\right)$$

Jumps “up” from a Z_j to a Z_i ($j > i$) are also possible. The number j_{up} of such jumps is restricted by t and by the maximum number of jumps “down” (from the sets Z_j to their complements S_j). Thus, we get:

$$j_{up} \leq t \cdot j_{down} = O\left(\frac{n \cdot \log(n)}{\sqrt{m(n)}}\right)$$

The running time for this case, as well as the overall running time, are bounded by $O\left(\frac{n \cdot \log(n)}{\sqrt{m(n)}}\right)$. We can choose any $m(n) = o(n)$. If we choose $m(n) = \frac{n}{\log^*(n)}$, we get $O(\sqrt{n \cdot \log^*(n)} \cdot \log(n))$.

Now we shall provide a sketch of the proof for the lower bound. Due to the birthday problem we know that it is expectedly sufficient to sample $\Theta(\sqrt{n})$ vertices non-adaptively until the selection contains a duplicate vertex. Consider a regular tournament G , that is, a tournament with a regular score vector. In each step one of $\frac{n-1}{2}$ vertices is chosen with equal probability. If in each step the same vertices were concerned (which, of course, is not very likely), the expected running time – until one vertex is hit for the second time – would be $\Theta(\sqrt{n})$. In case that the set of vertices changes the running time will not be smaller. $\square$

5 Conclusions

We have shown that a random walk on any tournament on n vertices finds either a sink or a 3-cycle in expected time $O\left(\sqrt{n} \log n \sqrt{\log^* n}\right)$. This gives a generic algorithm for solving an instance of LS_t in time $O^*(2^{t/2})$ ($n = 2^t$). Unfortunately, the computational model, which we use, is rather strong as it allows for uniform random sampling the set of all local moves from any given vertex. Therefore, the main open question is to find a sublinear (in n) algorithm that works in the standard computational model (in which an adversary may pick and return a local move from the set of all possibilities).

There might be better algorithms for finding a witness than the one stated in section 4. If the directions of the edges in the tournament were chosen randomly, a constant fraction of all sets of three vertices would consist of directed 3-cycle and an algorithm that simply samples three vertices in a round and checks if they form a 3-cycle would find a 3-cycle in constant time with high probability.

Of course this algorithm will fail or perform badly if there are no or very few directed 3-cycles in the graph. But in these cases the algorithm of section 4 should work well so that the two algorithms might complement each other. Hence, a “combined algorithm” running both algorithms in parallel might be better than either of them alone.

References

1. Camion, P.: Chemins et circuits hamiltoniens dans les graphes complets. *Comptes Rendus de la Académie des Sciences de Paris* 249, 2151–2152 (1959)
2. Czumaj, A., Sohler, C.: Sublinear-time Algorithms. *Bulletin of the EATCS* 89, 23–47 (2006)
3. Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: How Easy is Local Search? *J. Comput. Syst. Sci.* 37(1), 79–100 (1988)
4. Moon, J.W.: *Topics on Tournaments*. Holt, Rinehart, and Winston, New York (1968)
5. Reid, K.B., Beineke, L.W.: Tournaments. In: Beineke, L.W., Wilson, R.J. (eds.) *Selected Topics in Graph Theory*, pp. 169–204. Academic Press, London (1978)
6. Rubinfeld, R.: Sublinear time algorithms (2006),
iop <http://people.csail.mit.edu/ronitt/papers/index.html>

Classification of a Class of Counting Problems Using Holographic Reductions

Michael Kowalczyk

Department of Mathematics and Computer Science
Northern Michigan University, Marquette, MI 49855
mkowalcz@nmu.edu

Abstract. The purpose of this work is to prove a generalization of the dichotomy theorem from [6], extending that result to a larger class of counting problems. This is achieved through the use of interpolation and holographic reductions. We also use holographic reductions to establish a close connection between a class of problems which are solvable using Fibonacci gates and the class of problems which can be solved by applying a particular kind of counting argument.

Keywords: Fibonacci gates, holographic algorithms, holographic reduction, interpolation.

1 Introduction

The complexity class $\#P$, first introduced by Valiant [24], encompasses a diverse spread of counting problems. Some of these problems (such as counting perfect matchings in a planar graph) can be solved efficiently [22,19,20], but others (such as counting all, not necessarily perfect, matchings in a planar graph) have thwarted all attempts to find an efficient algorithm or to prove intractability. There has been much effort in recent years to further investigate and clarify the structure of $\#P$ by finding progressively larger classes of problems within $\#P$ for which each problem within that class can be proved to be either in P or $\#P$ -complete [13,14,16,2,6,9].

For example, let $G(V, E)$ be an undirected graph, and consider the problem of counting how many cuts of G have an even number of edges crossing the cut. One way to formulate this problem is to consider all $\{0, 1\}$ -assignments to the vertices of G and assign functions called *signatures* to every edge of G . In this case, we will assign the signature F with truth table $(1, -1, -1, 1)$ to all edges. Now let $Val(G) = \sum_{\sigma} \prod_{(x,y) \in E} F(\sigma(x), \sigma(y))$, where the sum is over all possible $\{0, 1\}$ -assignments to the vertices. Then $Val(G) = a - b$ where a is the number of even-sized cuts of G and b is the number of odd-sized cuts. Since $a + b = 2^{|E|}$, it is clear that the problem of calculating $Val(G)$ is equivalent to that of calculating a . Such problems can be generalized with the notion of H -Homomorphisms [13,14,16,2,17,18,21,15]. For example, one can consider a fixed undirected weighted graph $H = (V', E')$ and the problem is to count the sum weight of all homomorphisms of the input graph G to H . The problem of counting

even-sized cuts above can be modeled as an H -Homomorphism counting problem where the adjacency matrix of H is given by $A = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. A dichotomy theorem for all symmetric real matrices A is given by Goldberg, Grohe, Jerrum, and Thurley in [16], and a dichotomy theorem for all symmetric complex matrices is given by Cai, Chen, and Lu in [5].

A related and more general setting is counting constraint satisfaction problems ($\#CSP$). A problem instance consists of a set of variables, a finite set of values D they can take on, a finite set of constraints on the variables, and the goal of computing how many settings of the variables satisfy all of the constraints. This can be generalized to $\#Weighted CSP$ which allows for different weight assignments to the constraints. We can understand $\#Weighted CSP$ in terms of a bipartite graph with signatures on the vertices and assignments from D to the edges. Each vertex on the left side of the graph represents a variable and has an all-equal signature, whereas every vertex on the right side represents a constraint. There is an edge from each constraint vertex to every variable in its scope, and each constraint vertex takes on a signature that mirrors the requirements for the corresponding constraint. Then the counting problem can be expressed as a sum of products, where the sum is over all D -edge assignments and the product is over all signatures. Much work has been done in this area and dichotomy results are known for different variants of $\#CSP$ [1,2,10,9,12,13,14,23].

We can generalize both the edge-assignment and vertex-assignment viewpoints simultaneously by associating signatures to both vertices *and* edges and taking the sum over all possible D -assignments to both ends of each edge of an undirected graph G . This can be equivalently stated by restructuring G as a bipartite graph G' where the vertices on the left side all have degree 2 (these vertices correspond to the edges of G , whereas the right-side vertices of G' correspond to the vertices of G). The D -assignments are made to the edges of G' , and signatures are assigned to the vertices. Thus the study of edge signatures, simultaneous vertex and edge signatures, and $\#Weighted CSP$ are all subsumed into the study of vertex signatures and edge assignments in a bipartite graph, which is the view that we will adopt in this paper.

Our study of counting problems closely involves the study of holographic algorithms and holographic reductions [3,6,4,26,25,27]. The theory of holographic algorithms allows us to show that certain problems are computable in polynomial time, whereas the theory of holographic reductions allows us to draw connections between the complexity of problems which at first seem to be quite distinct. The usual framework for considering counting problems in the context of holographic algorithms is quite expressive. Signatures play an important role in the theory, and while some work has been done on *unsymmetric signatures*, we continue the study of *symmetric signatures* in this paper, for which much more is already known. Working with symmetric signatures will also simplify our notation somewhat (in the case of Boolean edge assignments, a symmetric signature depends only on the Hamming weight of its inputs).

In this paper we will consider bipartite graphs $G = (V, E)$ with left degree 2 and right degree 3. Each problem consists of a signature $[x_0, x_1, x_2]$ which is

applied to each vertex on the left side of the graph and a signature $[y_0, y_1, y_2, y_3]$ applied to each vertex on the right, where $x_i, y_j \in \{0, 1, -1\}$. The problem is to compute the sum of the products of all signatures within the graph under all possible $\{0, 1\}$ -assignments to the edges. We will prove a dichotomy theorem for this class of problems. In particular, we will show that for each setting of the x_i and y_j variables, the problem is either in P or #P-complete (we also show this for the case where we restrict to planar graphs). In [6], a dichotomy theorem is achieved in this setting for the case where $x_i, y_j \in \{0, 1\}$. In that paper, the hardness results were achieved uniformly by using an interpolation technique. In this paper, we show that the same technique can be applied uniformly to obtain the required hardness results, although a new gadget will be required in order for the interpolation to work in all cases. We will also study an interesting new tool known as *Fibonacci gates* [6], and give an alternate characterization for a class of problems that can be solved with them.

2 Definitions and Background

Let $G = (V, E)$ be an undirected graph and let $d(v)$ denote the degree of vertex $v \in V$. Let $\mathbf{F}$ be a fixed field and let the vertices $v \in V$ be labeled with functions $F_v \in \mathcal{F}$ where $F_v : \{0, 1, \dots, d(v)\} \rightarrow \mathbf{F}$. Then the pair $\Omega = (G, \mathcal{F})$ is called a (symmetric) *signature grid*. Given an assignment $\sigma : E \rightarrow \{0, 1\}$, the valuation at $v \in V$ under σ is defined to be $F_v(\sum_{e \in E_v} \sigma(e))$ where E_v is the set of edges incident on v . The value of Ω under σ is the product of valuations over all vertices $v \in V$, and the value of Ω (also known as Holant_Ω) is the sum of the value of Ω under all possible assignments σ . In other words,

$$\text{Holant}_\Omega = \sum_{\sigma} \prod_{v \in V} F_v \left(\sum_{e \in E_v} \sigma(e) \right),$$

where the outer sum is over all possible $\{0, 1\}$ -assignments to the edge set E . We can then view Ω as an instance of a counting problem, which is to compute Holant_Ω . For any vertex $v \in V$, the vector $[F_v(0), F_v(1), \dots, F_v(d(v))]$ is known as its *symmetric signature*. Although we only consider symmetric signatures in this paper, general (i.e. not necessarily symmetric) signature notation will also be useful. This consists of the entire truth table of the signature written as a $2^{d(v)}$ -length vector enclosed in parentheses. Thus, a symmetric signature written in general signature notation takes the form $(F_v(B(0)), F_v(B(1)), \dots, F_v(B(2^{d(v)} - 1)))$, where $B(x)$ is the sum of the digits of x when expressed in binary. In this paper a tilde above a symmetric signature denotes its general signature equivalent as a vector.

If $\mathcal{G}$ and $\mathcal{R}$ are sets of signatures, then the notation $\#\mathcal{G}|\mathcal{R}$ denotes the counting problem whose instances are the signature grids $(G, \mathcal{F})$ where G is bipartite and each vertex on the left side of G is assigned a signature from $\mathcal{G}$ (called the *generators*, whose signatures are written as column vectors) and each vertex on the right side of G a signature from $\mathcal{R}$ (called the *recognizers*, whose signatures are

written as row vectors). In the case where $\mathcal{G} = \{g\}$ and $\mathcal{R} = \{r\}$ are singletons, we allow the alternate notation $\#g|r$ for the counting problem.

We say that $\#\mathcal{G}|\mathcal{R}$ has a holographic reduction to $\#\mathcal{G}'|\mathcal{R}'$ if there is a basis $T \in GL_2(\mathbb{C})$ such that for all $G \in \mathcal{G}$ and $R \in \mathcal{R}$ there exist $G' \in \mathcal{G}'$ and $R' \in \mathcal{R}'$ such that $G' = T^{\otimes g}G$ and $R'T^{\otimes r} = R$ where g and r are the arity of G and R respectively (in these equations we are using general signature notation). Note that this particular definition of a holographic reduction is invertible (other variants exist). This leads us to the Holant Theorem, first discovered by Valiant [25], which is of central importance.

Theorem 1 (Holant Theorem). *Suppose there is a holographic reduction from $\#\mathcal{G}|\mathcal{R}$ to $\#\mathcal{G}'|\mathcal{R}'$ which induces a mapping of signature grid Ω to Ω' . Then $\text{Holant}_{\Omega} = \text{Holant}_{\Omega'}$.*

Most signatures $[x_0, x_1, x_2, x_3]$ can be transformed via holographic reduction to the form $[1, 0, 0, 1]$ or $[1, 1, 0, 0]$, but this is not the case for *degenerate signatures*. We call a symmetric signature $[x_0, x_1, x_2, \dots, x_n]$ a degenerate signature if the $2 \times n$ matrix $\begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_{n-1} \\ x_1 & x_2 & x_3 & \cdots & x_n \end{bmatrix}$ does not have rank 2.

Fibonacci gates are a new tool in the theory of holographic algorithms, introduced in [6]. Essentially, Fibonacci gates offer a recursive means of computing the holant of signature grids where every signature is a *Fibonacci signature*. A Fibonacci signature is a symmetric signature $[f_0, f_1, \dots, f_n]$ that satisfies the relation $f_{k+2} = f_{k+1} + f_k$ for all $k \in \{0, 1, \dots, n-2\}$. Once holographic reductions are taken into account, the following characterization can be made.

Theorem 2. *A set of symmetric generators $\mathcal{G} = \{G_1, G_2, \dots, G_s\}$ and symmetric recognizers $\mathcal{R} = \{R_1, R_2, \dots, R_t\}$ are all simultaneously realizable as Fibonacci gates after a holographic reduction under some basis $T \in GL_2(\mathbb{C})$ iff there exist three constants a, b , and c such that $b^2 - 4ac \neq 0$ and the following two conditions are satisfied:*

1. *For any $[x_1, x_2, \dots, x_{g_j}] \in \mathcal{G}$ and any $k \in \{0, 1, \dots, g_j - 2\}$, $cx_k - bx_{k+1} + ax_{k+2} = 0$.*
2. *For any $[y_1, y_2, \dots, y_{r_i}] \in \mathcal{R}$ and any $k \in \{0, 1, \dots, r_i - 2\}$, $ay_k + by_{k+1} + cy_{k+2} = 0$.*

Proof. See [6].

When working with signature grids, it can be useful to build up larger graph fragments from smaller ones in order to effectively “simulate” new signatures. For example, suppose we are working in the setting $\#[-1, 0, 1][0, 0, 1, 1]$. Although the signature $[-1, 0, 1]$ is the only generator available for the degree 2 vertices, we can build up a graph fragment like the one in Figure 1(d). From the perspective of the rest of the signature grid, this subgraph effectively acts like a generator with signature $[1, -1, -1]$. This brings us to the definition of an $\mathcal{F}$ -gate [6], which is almost the same as that of a signature grid.

Let $H = (V, E, D)$ be an undirected graph where D is a set of “dangling edges” (only one vertex incident to each dangling edge is in V). Let $\mathcal{F}$ be a set of signatures from which the vertices V are labeled. Then the pair $\Gamma = (H, \mathcal{F})$ is called an $\mathcal{F}$ -gate. If the edges E are denoted as $1, 2, \dots, m$ and the dangling edges D as $m + 1, m + 2, \dots, m + n$, then the signature of the $\mathcal{F}$ -gate Γ can be defined as the following function, where $(y_1, y_2, \dots, y_n)$ is a $\{0, 1\}$ -assignment to the dangling edges and $H(x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_n)$ is the value of Γ when viewed as a signature grid:

$$\Gamma(y_1, y_2, \dots, y_n) = \sum_{x_1 x_2 \dots x_m \in \{0,1\}^m} H(x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_n) .$$

Note that in general the signature of an $\mathcal{F}$ -gate need not be symmetric, even if all signatures in $\mathcal{F}$ are symmetric.

Interpolation, as a method to prove hardness of counting problems, was first given by Valiant [24]. This technique was later expanded upon by Dyer, Greenhill, and Vadhan [14,23]. We will employ a version of this powerful technique as proposed in [6]. Suppose we want to show that $\#[w, x, z][y_0, y_1, y_2, y_3]$ is $\#P$ -hard. The first observation is that given a signature grid Ω , we can replace each generator vertex with an $\mathcal{F}$ -gate H (where $\mathcal{F} = \{[w, x, z], [y_0, y_1, y_2, y_3]\}$), and we will still have a problem instance of $\#[w, x, z][y_0, y_1, y_2, y_3]$, as long as the $\mathcal{F}$ -gate maintains the 2-3 regular bipartite structure of the graph. Nevertheless, we can think of H as simulating the action of some generator $[w_1, x_1, z_1]$. In fact, we can try many different $\mathcal{F}$ -gates in place of the original generator vertices. In [6], a recursive gadget construction is used to produce an infinite set of $\mathcal{F}$ -gates for this purpose, and we will use the same technique here. Let N_0 be a single generator vertex, and define $\mathcal{F}$ -gate N_s recursively using N_{s-1} and a gadget (as shown in Figure 1). Building up bigger $\mathcal{F}$ -gates with the same gadget makes it possible to describe the recurrence in terms of a matrix A , so that the signature of N_s is $[w_s, x_s, z_s]^T = A^s[w, x, z]^T$. At this point, it can already be observed that if $\#[w_s, x_s, z_s][y_0, y_1, y_2, y_3]$ is $\#P$ -hard, then so is $\#[w, x, z][y_0, y_1, y_2, y_3]$.

The next step is to note that we can write out the holant for each of these graphs as a linear sum of the same $\binom{n+2}{2}$ constants $c_{i,j,k}$, where n is the number of generator vertices in Ω . Let Ω_s be Ω with N_s substituted for the generators, and let $c_{i,j,k}$ be a sum over products of all recognizer values, where the sum is over all assignments to the edges for which the number of generators with 0, 1, and 2 incoming edges with 1s are i, j , and k respectively. Then

$$\text{Holant}_{\Omega_s} = \sum_{\sigma} \prod_{v \in V} F_v \left(\sum_{e \in E_v} \sigma(e) \right) = \sum_{i+j+k=n} c_{i,j,k} w^i x^j z^k . \tag{1}$$

Each $c_{i,j,k}$ really comes from applying the distributive property to the definition of the holant when the different $w^i x^j z^k$ terms are extracted, but it is important to note that the same $c_{i,j,k}$ constants appear for each Ω_s so that we can frame this as a linear system and solve for these constants. The remaining step is to find sufficient conditions to confirm that such a system of equations has full rank

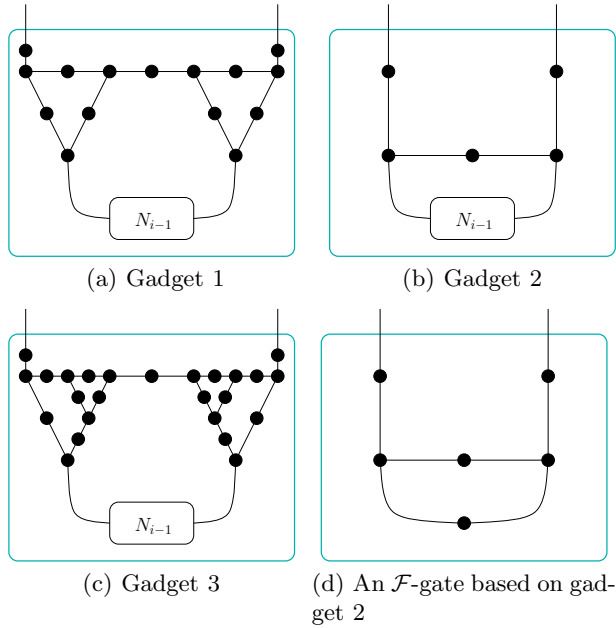


Fig. 1. Three gadgets and an $\mathcal{F}$ -gate

so we can determine the $c_{i,j,k}$ constants exactly. Then as long as there exist *any* constants w' , x' , and z' for which the problem $\#[w', x', z']|[y_0, y_1, y_2, y_3]$ is $\#P$ -hard, the reduction is complete and we conclude that $\#[w, x, z]|[y_0, y_1, y_2, y_3]$ is also $\#P$ -hard. In fact, for every non-degenerate signature $[y_0, y_1, y_2, y_3]$ there exist x_0 , x_1 , and x_2 such that $\#[x_0, x_1, x_2]|[y_0, y_1, y_2, y_3]$ is $\#P$ -complete. Sufficient conditions for interpolation to work are summarized in the following theorem, which follows from [6]:

Theorem 3. *Let $\#[x_0, x_1, x_2]|[y_0, y_1, y_2, y_3]$ be a counting problem where $x_i, y_j \in \mathbb{Q}$ and $[y_0, y_1, y_2, y_3]$ is non-degenerate. Let D be a gadget that admits the above recursive construction, and let A be the recurrence matrix for D (which transforms the signature of N_{i-1} to the signature of N_i). Suppose that $\det(A) \neq 0$, $[x_0, x_1, x_2]$ is not orthogonal to any row eigenvector of A , and the characteristic polynomial of A is irreducible over $\mathbb{Q}$ and not of the form $x^3 + c$. Then $\#[x_0, x_1, x_2]|[y_0, y_1, y_2, y_3]$ is $\#P$ -complete. Furthermore, if D is planar, then $\#[x_0, x_1, x_2]|[y_0, y_1, y_2, y_3]$ is $\#P$ -complete when restricted to planar graphs.*

3 A Characterization of Fibonacci Gates

In [6], different techniques are used to place individual problems in P . One might roughly categorize these as: connectivity arguments, counting arguments, Fibonacci gates, and degenerate cases. We will try to give a slightly more uniform

treatment by showing that with the aid of holographic reductions, any problem that we can solve in the bipartite setting with Fibonacci gates can also be solved using a connectivity argument and conversely. On the other hand, we will also encounter other problems which will require a different technique altogether (Lemma 2). Note that the conditions in the following lemma are precisely the same as those used in Theorem 2.

Lemma 1. *Let $\#\mathcal{G}|\mathcal{R}$ be a problem consisting of symmetric generators $\mathcal{G} = \{G_1, G_2, \dots, G_s\}$ and symmetric recognizers $\mathcal{R} = \{R_1, R_2, \dots, R_t\}$. Then $\#\mathcal{G}|\mathcal{R}$ can be reduced via holographic reduction to a problem $\#\mathcal{G}'|\mathcal{R}'$ with symmetric generators of the form $G'_i = [a_i, 0, 0, \dots, 0, b_i]$ and symmetric recognizers of the form $R'_i = [c_i, 0, 0, \dots, 0, d_i]$ if and only if there are constants a , b , and c such that $b^2 - 4ac \neq 0$ and:*

1. *For any $[x_1, x_2, \dots, x_{g_j}] \in \mathcal{G}$ and any $k \in \{0, 1, \dots, g_j - 2\}$, $cx_k - bx_{k+1} + ax_{k+2} = 0$.*
2. *For any $[y_1, y_2, \dots, y_{r_i}] \in \mathcal{R}$ and any $k \in \{0, 1, \dots, r_i - 2\}$, $ay_k + by_{k+1} + cy_{k+2} = 0$.*

Proof. Let $\#\mathcal{G}'|\mathcal{R}'$ be a signature grid with symmetric generators of the form $G'_i = [a_i, 0, 0, \dots, 0, b_i]$, symmetric recognizers of the form $R'_i = [c_i, 0, 0, \dots, 0, d_i]$, and let $T = \begin{bmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \end{bmatrix}$ be an invertible matrix. Let g_i and r_i denote the arity of G'_i and R'_i , respectively. Applying a holographic reduction to $\#\mathcal{G}'|\mathcal{R}'$ using T , we have $\widetilde{G}_i = T^{\otimes g_i} \widetilde{G}'_i$, so in symmetric notation $G_i = [a_i \alpha_1^{g_i} + b_i \beta_1^{g_i}, a_i \alpha_1^{g_i-1} \alpha_2 + b_i \beta_1^{g_i-1} \beta_2, \dots, a_i \alpha_2^{g_i} + b_i \beta_2^{g_i}]$, that is, the element at zero-based index j in the symmetric signature of G_i is $a_i \alpha_1^{g_i-j} \alpha_2^j + b_i \beta_1^{g_i-j} \beta_2^j$. Since $T^{-1} = \frac{1}{d} \begin{bmatrix} \beta_2 & -\beta_1 \\ -\alpha_2 & \alpha_1 \end{bmatrix}$

where $d = \det(T)$, we find $\widetilde{R}_i = \widetilde{R}'_i (T^{-1})^{\otimes r_i}$ to have $c_i (\beta_2/d)^{r_i-j} (-\beta_1/d)^j + d_i (-\alpha_2/d)^{r_i-j} (\alpha_1/d)^j$ as its element at index j . Interpreting the signatures of G_i and R_i as second order linear homogeneous recurrence relations, we see that the roots of the characteristic polynomials of the recurrences are $\gamma_1 := \alpha_2/\alpha_1$ and $\gamma_2 := \beta_2/\beta_1$ for G_i , and for R_i they are $-\beta_1/\beta_2 = -\gamma_2^{-1}$ and $-\alpha_1/\alpha_2 = -\gamma_1^{-1}$, regardless of i in both cases. The associated characteristic polynomials for the generator and recognizer recurrences are then $x^2 - (\gamma_1 + \gamma_2)x + \gamma_1\gamma_2 = 0$ and $\gamma_1\gamma_2x^2 + (\gamma_1 + \gamma_2)x + 1 = 0$ respectively, thus we have the relation $cx_k - bx_{k+1} + ax_{k+2} = 0$ for each generator and $ax_k + bx_{k+1} + cx_{k+2} = 0$ for each recognizer where $a = 1$, $b = \gamma_1 + \gamma_2$, and $c = \gamma_1\gamma_2$. Note that $b^2 - 4ac = (\gamma_1 + \gamma_2)^2 - 4\gamma_1\gamma_2 = (\gamma_1 - \gamma_2)^2 \neq 0$ as required, since $\det(T) \neq 0$.

Conversely, suppose $\#\mathcal{G}|\mathcal{R}$ is a problem with symmetric generators $G_1, G_2, \dots, G_s$ and symmetric recognizers $R_1, R_2, \dots, R_t$, and there exist a, b, c with $b^2 - 4ac \neq 0$ such that for any generator $[x_0, x_1, \dots, x_g] = G \in \{G_1, \dots, G_s\}$, we have $cx_k - bx_{k+1} + ax_{k+2} = 0$ and for any recognizer $[x_0, x_1, \dots, x_r] = R \in \{R_1, \dots, R_t\}$, we have $ax_k + bx_{k+1} + cx_{k+2} = 0$. Since $b^2 - 4ac \neq 0$, the roots of the characteristic polynomials are distinct, and we can write the

generator signature of G_i such that the element at index j is $a_i \alpha_1^{g_i-j} \alpha_2^j + b_i \beta_1^{g_i-j} \beta_2^j$, for some fixed a_i and b_i . Similarly, we can have recognizer R_i take the value $c_i d^{-r_i} (\beta_2)^{r_i-j} (-\beta_1)^j + d_i d^{-r_i} (-\alpha_2)^{r_i-j} (\alpha_1)^j$ at index j , where $d = \det(T)$ as before (note that $d \neq 0$ because $b^2 - 4ac \neq 0$). The constants a_i and b_i in the case of generators and c_i and d_i in the case of recognizers are uniquely determined by the first two values of its signature. Now applying the same holographic reduction as before, we get symmetric signatures with the desired form.

4 Classification of Problems

The goal of this section is to prove a dichotomy theorem for problems of the form $\#[x_0, x_1, x_2][y_0, y_1, y_2, y_3]$, where $x_i, y_j \in \{0, 1, -1\}$, placing each problem in one of the following three categories.

1. The problem is in P.
2. The problem is $\#P$ -complete in general, but in P when restricted to planar graphs.
3. The problem is $\#P$ -complete, even when restricted to planar graphs.

We start with a few observations regarding relationships between different problems.

1. We can reverse the order of both the generator and recognizer signatures of any problem, and this can be justified by a holographic reduction using the matrix $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ (or switch the roles of the 0s and 1s in the assignments to the edges, as pointed out in [6]).
2. Multiplying each entry in a generator signature by -1 has the effect of multiplying the value of the signature grid by $(-1)^s$ where s is the number of generators (and similarly for recognizers).
3. The problems $\#[x_0, x_1, x_2][y_0, y_1, y_2, y_3]$ and $\#[x_0, -x_1, x_2][y_0, -y_1, y_2, -y_3]$ are equivalent by a holographic reduction using the matrix $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

Now define the equivalence relation $\sim$ so that two problems are considered equivalent under $\sim$ if and only if one can be reduced to the other via some combination of one or more of the above 3 transformations, so that members of each equivalence class of $\sim$ share the same complexity. This equivalence relation will simplify our discussion.

4.1 Tractable Cases

Here we list which problems are in P and how they can be solved efficiently. We first consider the degenerate signatures. In our case, these consist of problems that have the generator signatures $[0, 0, 0]$, $[0, 0, x]$, $[x, 0, 0]$, $[x, x, x]$, or $[x, -x, x]$, as well as any problems with recognizer signatures $[0, 0, 0, 0]$, $[0, 0, 0, y]$, $[y, 0, 0, 0]$, $[y, y, y, y]$, or $[y, -y, y, -y]$, where $x, y \in \{1, -1\}$. Both $\#[0, 0, 0][y_0, y_1, y_2, y_3]$

and $\#[x_0, x_1, x_2][0, 0, 0, 0]$ trivially evaluate to zero, and problems of the form $\#[0, 0, x][y_0, y_1, y_2, y_3]$ and $\#[x_0, x_1, x_2][0, 0, 0, y]$ evaluate to $x^s y_3^t$ and $x_2^s y^t$ respectively, where s is the number of generators and t is the number of recognizers in the signature grid. Also, $\#[x, x, x][y_0, y_1, y_2, y_3]$ and $\#[x_0, x_1, x_2][y, y, y, y]$ evaluate to $x^s(y_0 + 3y_1 + 3y_2 + y_3)^t$ and $(x_0 + 2x_1 + x_2)^s y^t$ respectively. Since $\#[x, -x, x][y_0, y_1, y_2, y_3] \sim \#[x, x, x][y_0, -y_1, y_2, -y_3]$, $\#[x_0, x_1, x_2][y, 0, 0, 0] \sim \#[x_2, x_1, x_0][0, 0, 0, y]$, $\#[x_0, x_1, x_2][y, -y, y, -y] \sim \#[x_0, -x_1, x_2][y, y, y, y]$, and $\#[x, 0, 0][y_0, y_1, y_2, y_3] \sim \#[0, 0, x][y_3, y_2, y_1, y_0]$, we conclude that all degenerate cases can be solved in polynomial time.

As pointed out in [6], some problems always evaluate to zero due to a counting argument. For example, generators of the form $[x_0, x_1, 0]$ effectively require that at most half of the edges have nonzero assignments, and recognizers of the form $[0, 0, y_2, y_3]$ demand that at least two-thirds of the edges have nonzero assignments. These requirements are incompatible so any problem instance of the form $\#[x_0, x_1, 0][0, 0, y_2, y_3]$, has a signature grid with value zero.

Problems where recognizers have the form $[y_0, 0, 0, y_3]$ and generators have the form $[x_0, 0, x_2]$ or $[0, x_1, 0]$ can be solved in polynomial time with a connectivity argument [6]. That is, once an assignment to an edge has been made, all edge assignments to adjacent edges become determined, and the entire connected component has its edge assignments determined as a result (if a consistent assignment to that connected component even exists). Once the calculation has been made for each connected component, the product of these is the value of the signature grid. Given Lemma 1, this technique becomes widely applicable to the problems we are considering. Furthermore, a connectivity argument can also be carried out for problems of the form $\#[x, 0, -x][y, 0, y, 0]$, $\#[x, 0, -x][0, y, 0, y]$, and $\#[x, 0, x][y, z, -y, -z]$. Using the reduction $R = [c, 0, 0, 0, 0, 0, d] \cdot T^{\otimes 3}$, $G = (T^{-1})^{\otimes 2} \cdot [0, a, a, 0]^T$, these problems reduce to $\#[0, a, 0][c, 0, 0, d]$ for some $a, c, d \in \mathbb{C}$ using the bases $T_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, $T_2 = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$, and $T_3 = \begin{bmatrix} 1 & i \\ 1 & -i \end{bmatrix}$ respectively.

One final class of problems remain that are computable in P. These are the problems with generators of the form $[x, x, -x]$ or $[x, -x, -x]$ and recognizers of the form $[y, 0, 0, z]$, $[y, 0, y, 0]$, or $[0, y, 0, y]$ where $x, y, z \in \{1, -1\}$. These are handled by the following lemma, which follows from [8].

Lemma 2. *If Ω is a signature grid that consists only of the signatures $[1, 1, -1]$, $[1, -1, -1]$, $[0, 1, 0, 1]$, $[1, 0, 1, 0]$, $[1, 0, 0, -1]$, and $[1, 0, 0, 1]$, then Holant_Ω can be computed in polynomial time.*

4.2 Intractable for Nonplanar Graphs but Tractable for Planar Graphs

Some problems are $\#P$ -complete in general but are in P when restricted to planar graphs. For example, it is known that the problems $\#[1, 0, 1][0, 1, 0, 0]$, $\#[1, 0, 1][0, 1, 1, 0]$, and $\#[0, 1, 0][0, 1, 1, 0]$ fall in this category [6]. Then we get $\#[1, 0, 1][0, 1, 0, 0] \sim \#[x, 0, x][0, y, 0, 0]$, $\#[1, 0, 1][0, 1, 1, 0] \sim \#[x, 0, x][$

$[0, y, z, 0]$, and $\#[0, 1, 0][0, 1, 1, 0] \sim \#[0, x, 0][0, y, z, 0]$ for all $x, y, z \in \{1, -1\}$. It turns out that these are the only $\{0, 1, -1\}$ signatures which are $\#P$ -complete in general but are in P when restricted to planar graphs, as we will verify shortly.

4.3 Intractable Even for Planar Graphs

There are 48 problems which we show to be $\#P$ -complete, even in the planar case, by using the general strategy of interpolation as in Theorem 3. In each case, one of three general gadgets is applied to the problem, a recurrence matrix is calculated, and sufficient conditions on the matrix are verified to be met. This will be enough to guarantee $\#P$ completeness (this proves it for the planar case too, since the gadgets are planar). The list of problems, which gadgets were applied, and the resulting irreducible characteristic polynomials are in the appendix (problems with only Boolean signatures are omitted). For most problems, at least one of the two gadgets given in [6] was sufficient for interpolation, but in a few cases a new gadget (see Figure 1(c)) was needed. Since the polynomials have integer coefficients, they can be shown to be irreducible over $\mathbb{Q}$ via Gauss's lemma by checking that the roots are not integral.

To illustrate the process, we will prove that $\#[-1, -1, 1][1, 1, 1]$ is $\#P$ -hard. For this particular problem, it turns out that both gadget 1 and gadget 2 do not meet the conditions of Theorem 3, so we will try gadget 3. We calculate

that the recurrence matrix is $A = \begin{bmatrix} -9216 & -36864 & -4096 \\ 15360 & -8192 & -12288 \\ 7168 & 20480 & -4096 \end{bmatrix}$, which means that

an $\mathcal{F}$ -gate built using s iterations of gadget 3 will have a signature given by $A^s \cdot [-1, -1, 1]^T$ (note that symmetry of the $\mathcal{F}$ -gate's signature is inherited from the gadget's symmetry). Now we verify that the technical conditions hold. The characteristic polynomial of A is $f(x) = 3229815406592 + 994050048x + 21504x^2 + x^3$, so clearly $\det(A) \neq 0$ and $f(x)$ is not of the form $x^3 + c$. There is only one real root of $f(x)$, and it is at $x \approx -3467.28$. Since the polynomial has integer coefficients and no integer roots, we conclude by Gauss's lemma that $f(x)$ is irreducible over the rationals. Finally, we need to verify that $[-1, -1, 1]$ is not orthogonal to any row eigenvector of A . Suppose u is a row eigenvector of A and u is orthogonal to $[-1, -1, 1]$, so that $u = [a, b, a + b]$ for some a and b and $uA = \lambda u$ where $\lambda \neq 0$. Then $\lambda u = uA = -2048[a - 11b, 8a - 6b, 4a + 8b]$, thus $-2048(4a + 8b) = \lambda(a + b) = \lambda a + \lambda b = -2048(a - 11b + 8a - 6b)$, which yields $a = 5b$. Then $\lambda[5b, b, 6b] = \lambda u = [-6b, 34b, 28b]$, from which we conclude that $b = 0$, $a = 0$, and no row eigenvector is orthogonal to $[-1, -1, 1]$. Gadget 3 is planar, so by Theorem 3, $\#[-1, -1, 1][1, 1, 1]$ is $\#P$ -hard, even for planar graphs. Meeting the technical conditions of the theorem verifies that we can build a linear system of full rank to solve for the constants $c_{i,j,k}$ in equation (1). If the $c_{i,j,k}$ constants are in hand, then one can solve any problem of the form $\#[x_0, x_1, x_2][1, 1, 1]$, but since $[-1, 1, 1, 1]$ is non-degenerate, there also exist x_0, x_1 , and x_2 such that $\#[x_0, x_1, x_2][1, 1, 1]$ is $\#P$ -hard, and this completes the reduction.

Table 1. Classification of problems where $x, y \in \{1, -1\}$

Recognizer	#P-hard	P when planar	Counting	Connectivity	Lemma 2
$[0, 1, 1, 1]$	$[0, x, x], [x, x, 0],$ $[x, 0, y], [x, x, -x],$ $[0, x, 0], [0, x, -x]$ ¹			$[x, -x, -x]$ $[x, -x, 0]$	
$[-1, 1, 1, 1]$	$[0, x, x], [x, x, 0],$ $[x, 0, y], [x, x, -x],$ $[0, x, 0], [0, x, -x]$			$[x, -x, -x]$ $[x, -x, 0]$	
$[1, 0, 1, 1]$	$[-x, x, x], [-x, 0, x],$ $[0, -x, x], [x, x, -x],$ $[x, x, 0], [0, x, 0]$			$[x, 0, x]$ $[0, x, x]$ $[x, -x, 0]$	
$[0, 0, 1, 1]$	$[0, x, x], [-x, x, x],$ $[x, 0, y], [0, x, -x],$ $[x, x, -x]$		$[0, x, 0]$ $[x, y, 0]$		
$[-1, 0, 1, 1]$	$[0, x, x], [-x, x, x],$ $[x, 0, x], [x, x, -x],$ $[x, x, 0], [0, x, 0]$			$[x, -x, 0]$ $[x, 0, -x]$ $[0, x, -x]$	
$[0, -1, 1, 1]$	$[0, x, x], [x, 0, x],$ $[0, x, -x], [x, x, -x],$ $[0, x, 0], [x, -x, 0]$ $[x, 0, -x]$ ²			$[x, x, 0]$ $[-x, x, x]$	
$[-1, -1, 1, 1]$	$[0, x, y], [x, y, 0]$			$[x, -x, -x]$ $[x, x, -x]$ $[0, x, 0]$ $[x, 0, y]$	
$[0, 1, 0, 1]$	$[0, x, y], [x, y, 0]$			$[x, 0, y]$ $[0, x, 0]$	$[-x, x, x]$ $[x, x, -x]$
$[1, 0, 0, 1]$	$[0, x, x], [x, x, 0]$ $[x, -x, 0], [0, x, -x]$ ³			$[x, 0, y]$ $[0, x, 0]$	$[-x, x, x]$ $[x, x, -x]$
$[0, -1, 0, 1]$	$[0, x, y], [x, y, 0]$			$[-x, x, x]$ $[x, x, -x]$ $[0, x, 0]$ $[x, 0, y]$	
$[0, 1, 1, 0]$	$[0, x, x], [x, x, 0],$ $[-x, x, x], [x, x, -x]$	$[0, x, 0]$ $[x, 0, x]$		$[x, -x, 0]$ $[0, x, -x]$ $[x, 0, -x]$	
$[0, 0, 1, 0]$	$[0, x, y], [x, x, -x],$ $[-x, x, x]$	$[x, 0, y]$	$[x, y, 0]$ $[0, x, 0]$		

¹ $\#[0, 1, 1, 1]$ reduces to $\#[0, -1, 1][0, 1, 1, 1]$ under basis $\begin{bmatrix} -1 & -1 \\ 0 & 1 \end{bmatrix}$.

² $\#[0, 1, 1][0, -1, 1, 1]$ reduces to $\#[0, -1, 1][0, -1, 1, 1]$ under basis $\begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}$, so the generators $[x, 0, -x]$ are also hard.

³ Cases $[x, -x, 0]$ and $[0, x, -x]$ are handled by the fact that $\#[0, -1, 1][0, 1, 1, 1]$ reduces to $\#[0, 1, -1][1, 0, 0, 1]$ under basis $\begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}$ and thus are hard.

4.4 Putting It All Together

Theorem 4. *All problems of the form $\#[x_0, x_1, x_2][y_0, y_1, y_2, y_3]$ where $x_i, y_j \in \{0, 1, -1\}$ are either 1) $\#P$ -complete in general but in P when restricted to planar graphs, 2) $\#P$ -complete even for planar graphs, or 3) in P .*

Proof. As we saw earlier, all degenerate cases are in P , so we need not consider any cases where the generator or recognizer are degenerate (this handles 9 generators and 9 recognizers). Under $\sim$, the 72 remaining recognizers fall into 12 equivalence classes: 6 of them have 8 members each (we will identify these by the representatives $[0, 1, 1, 1]$, $[-1, 1, 1, 1]$, $[1, 0, 1, 1]$, $[0, 0, 1, 1]$, $[-1, 0, 1, 1]$, and $[0, -1, 1, 1]$), and the other 6 have 4 members each (identified by the representatives $[-1, -1, 1, 1]$, $[0, 1, 0, 1]$, $[1, 0, 0, 1]$, $[0, -1, 0, 1]$, $[0, 1, 1, 0]$, and $[0, 0, 1, 0]$). To classify the complexity of all of these problems, it suffices to classify all 18 non-degenerate generators for each of these recognizer representatives. In the cases where the problem turns out to be in P , either a counting argument, connectivity argument, or Lemma 2 was applied. The problems that were tractable in the planar case but $\#P$ -hard in general were handled with holographic reductions as discussed above. Each $\#P$ -hard problem was either proved directly using interpolation or indirectly using a holographic reduction. The results are summarized in Table 1. Hardness reductions are listed in the appendix.

Acknowledgements

I would very much like to thank Jin-Yi Cai, Pinyan Lu, and Mingji Xia for helpful comments and discussions.

References

1. Bulatov, A.A., Dalmau, V.: Towards a Dichotomy Theorem for the Counting Constraint Satisfaction Problem. *Information and Computation* 205(5), 651–678 (2007)
2. Bulatov, A.A., Grohe, M.: The Complexity of Partition Functions. *Theoretical Computer Science* 348(2-3), 148–186 (2005)
3. Cai, J.-Y., Lu, P.: Holographic Algorithms: From Art to Science. In: *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pp. 401–410. ACM Press, New York (2007)
4. Cai, J.-Y., Lu, P.: On Symmetric Signatures in Holographic Algorithms. In: Thomas, W., Weil, P. (eds.) *STACS 2007*. LNCS, vol. 4393, pp. 429–440. Springer, Heidelberg (2007)
5. Cai, J.-Y., Chen, X., Lu, P.: Graph Homomorphisms with Complex Values: A Dichotomy Theorem. *Computing Research Repository*, arXiv:0903.4728v1 (2009)
6. Cai, J.-Y., Lu, P., Xia, M.: Holographic Algorithms by Fibonacci Gates and Holographic Reductions for Hardness. In: *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 644–653. IEEE Computer Society Press, Los Alamitos (2008)
7. Cai, J.-Y., Lu, P., Xia, M.: A Computational Approach to Proving Computational Complexity of Some Counting Problems. In: *Theory and Applications of Models of Computation: 6th International Conference (to appear, 2009)*

8. Cai, J.-Y., Lu, P., Xia, M.: Holant Problems and Counting CSP. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing (to appear, 2009)
9. Creignou, N., Hermann, M.: Complexity of Generalized Satisfiability Counting Problems. *Information and Computation* 125(1), 1–12 (1996)
10. Creignou, N., Khanna, S., Sudan, M.: Complexity Classifications of Boolean Constraint Satisfaction Problems. Society for Industrial and Applied Mathematics, Philadelphia (2001)
11. Dodson, C.T.J., Poston, T.: *Tensor Geometry*. Springer, New York (1991)
12. Dyer, M.E., Goldberg, L.A., Jerrum, M.: The Complexity of Weighted Boolean #CSP. Computing Research Repository, arXiv:0704.3683v2 (2008)
13. Dyer, M.E., Goldberg, L.A., Paterson, M.: On Counting Homomorphisms to Directed Acyclic Graphs. *Journal of the ACM* 54(6) (2007)
14. Dyer, M.E., Greenhill, C.S.: The Complexity of Counting Graph Homomorphisms. *Random Structures and Algorithms* 17(3-4), 260–289 (2000)
15. Freedman, M., Lovász, L., Schrijver, A.: Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society* 20, 37–51 (2007)
16. Goldberg, L.A., Grohe, M., Jerrum, M., Thurley, M.: A complexity dichotomy for partition functions with mixed signs. In: Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (to appear, 2009)
17. Goldberg, L.A., Kelk, S., Paterson, M.: The complexity of choosing an H-colouring (nearly) uniformly at random. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 53–62. ACM Press, New York (2002)
18. Hell, P., Nešetřil, J., Zhu, X.: Duality and polynomial testing of tree homomorphisms. *Transactions of the American Mathematical Society* 348(4), 1281–1297 (1996)
19. Kasteleyn, P.W.: The statistics of dimers on a lattice. *Physica* 27, 1209–1225 (1961)
20. Kasteleyn, P.W.: Graph Theory and Crystal Physics. In: *Graph Theory and Theoretical Physics*, pp. 43–110. Academic Press, London (1967)
21. Lovász, L.: Operations with structures. *Acta Mathematica Academiae Scientiarum Hungaricae* 18, 321–328 (1967)
22. Temperley, H.N.V., Fisher, M.E.: Dimer problem in statistical mechanics - an exact result. *Philosophical Magazine* 6, 1061–1063 (1961)
23. Vadhan, S.P.: The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM Journal on Computing* 31(2), 398–427 (2001)
24. Valiant, L.G.: The Complexity of Computing the Permanent. *Theoretical Computer Science* 8, 189–201 (1979)
25. Valiant, L.G.: Holographic Algorithms (Extended Abstract). In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 306–315. IEEE Computer Society Press, Los Alamitos (2004)
26. Valiant, L.G.: Quantum Circuits That Can Be Simulated Classically in Polynomial Time. *SIAM Journal on Computing* 31(4), 1229–1254 (2002)
27. Valiant, L.G.: Accidental Algorithms. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, pp. 509–517. IEEE Computer Society Press, Los Alamitos (2006)

Appendix

Problem	Gadget	Irreducible characteristic polynomial
$\#[-1, 0, 1][0, 1, 1, 1]$	1	$1 + 4X + 2X^2 + X^3$
$\#[-1, -1, 1][0, 1, 1, 1]$	2	$32 + 12X - 3X^2 + X^3$
$\#[0, 1, 1][-1, 1, 1, 1]$	1	$-1259712 + 1285956X - 11691X^2 + X^3$
$\#[1, 0, 1][-1, 1, 1, 1]$	3	$-69468160 + 1683456X - 4800X^2 + X^3$
$\#[-1, 0, 1][-1, 1, 1, 1]$	1	$4096 + 256X - 16X^2 + X^3$
$\#[0, -1, 1][-1, 1, 1, 1]$	1	$64 + 100X + 17X^2 + X^3$
$\#[-1, -1, 1][-1, 1, 1, 1]$	3	$3229815406592 + 994050048X + 21504X^2 + X^3$
$\#[1, 1, 0][-1, 1, 1, 1]$	2	$-48 + 40X - 3X^2 + X^3$
$\#[0, 1, 0][-1, 1, 1, 1]$	1	$-32768 + 1024X + 32X^2 + X^3$
$\#[-1, 1, 1][1, 0, 1, 1]$	2	$72 - 18X - 7X^2 + X^3$
$\#[-1, 0, 1][1, 0, 1, 1]$	2	$2 + X + X^2 + X^3$
$\#[0, -1, 1][1, 0, 1, 1]$	2	$2 + 5X - X^2 + X^3$
$\#[-1, -1, 1][1, 0, 1, 1]$	2	$24 + 2X + 5X^2 + X^3$
$\#[-1, 1, 1][0, 0, 1, 1]$	2	$-8 + 32X - 12X^2 + X^3$
$\#[-1, 0, 1][0, 0, 1, 1]$	2	$-1 + X + X^2 + X^3$
$\#[0, -1, 1][0, 0, 1, 1]$	2	$-1 + 4X - 2X^2 + X^3$
$\#[-1, -1, 1][0, 0, 1, 1]$	2	$-8 + 8X + X^3$
$\#[0, 1, 1][-1, 0, 1, 1]$	2	$-14 + 37X - 13X^2 + X^3$
$\#[-1, 1, 1][-1, 0, 1, 1]$	2	$-56 + 62X - 15X^2 + X^3$
$\#[1, 0, 1][-1, 0, 1, 1]$	2	$2 - X - 3X^2 + X^3$
$\#[-1, -1, 1][-1, 0, 1, 1]$	2	$56 + 26X - 3X^2 + X^3$
$\#[1, 1, 0][-1, 0, 1, 1]$	2	$-14 + X + 5X^2 + X^3$
$\#[0, 1, 0][-1, 0, 1, 1]$	2	$-4 + 4X + X^2 + X^3$
$\#[0, 1, 1][0, -1, 1, 1]$	2	$-8 + 11X - 2X^2 + X^3$
$\#[1, 0, 1][0, -1, 1, 1]$	2	$6 - X - 4X^2 + X^3$
$\#[0, -1, 1][0, -1, 1, 1]$	2	$-48 + 37X - 4X^2 + X^3$
$\#[-1, -1, 1][0, -1, 1, 1]$	2	$-288 + 36X + 13X^2 + X^3$
$\#[0, 1, 0][0, -1, 1, 1]$	2	$8 + 16X + 7X^2 + X^3$
$\#[-1, 1, 0][0, -1, 1, 1]$	2	$22 + 43X + 14X^2 + X^3$
$\#[0, 1, 1][-1, -1, 1, 1]$	2	$-40 + 26X - X^2 + X^3$
$\#[0, -1, 1][-1, -1, 1, 1]$	2	$-40 + 34X - 5X^2 + X^3$
$\#[0, 1, 1][0, -1, 0, 1]$	2	$-5 + 2X + 4X^2 + X^3$
$\#[1, 1, 0][0, -1, 0, 1]$	2	$-5 + 13X - 7X^2 + X^3$
$\#[-1, 1, 1][0, 1, 1, 0]$	2	$-56 + 24X + 4X^2 + X^3$
$\#[-1, 1, 1][0, 0, 1, 0]$	2	$-8 + 18X - 5X^2 + X^3$

Separating NE from Some Nonuniform Nondeterministic Complexity Classes

Bin Fu¹, Angsheng Li², and Liyu Zhang³

¹ Dept. of Computer Science, University of Texas - Pan American
TX 78539, USA

`binfu@cs.panam.edu`

² Institute of Software, Chinese Academy of Sciences, Beijing, P.R. China
`angsheng@gcl.iscas.ac.cn`

³ Department of Computer and Information Sciences, University of Texas at
Brownsville, Brownsville, TX, 78520, USA
`liyu.zhang@utb.edu`

Abstract. We investigate the question whether NE can be separated from the reduction closures of tally sets, sparse sets and NP. We show that (1) $NE \not\subseteq R_{n^{o(1)}-T}^{NP}(\text{TALLY})$; (2) $NE \not\subseteq R_m^{SN}(\text{SPARSE})$; and (3) $NE \not\subseteq P_{n^k-T}^{NP}/n^k$ for all $k \geq 1$. Result (3) extends a previous result by Mocas to nonuniform reductions. We also investigate how different an NE-hard set is from an NP-set. We show that for any NP subset A of a many-one-hard set H for NE, there exists another NP subset A' of H such that $A' \supseteq A$ and $A' - A$ is not of sub-exponential density.

1 Introduction

This paper continues a line of research that tries to separate nondeterministic complexity classes in a stronger sense, i.e., separating nondeterministic complexity classes from the *reduction closure* of classes with lower complexity. We focus on the class NE of nondeterministically exponential-time computable sets. Two most interesting but long standing open problems regarding NE are whether every NE-complete set is polynomial-time Turing reducible to an NP set and whether it is polynomial-time Turing reducible to a sparse set. The latter question is equivalent to whether every NE-complete set has polynomial-size circuits, since a set is polynomial-time Turing reducible to a sparse set if and only if it has polynomial-size circuits [1]. We show results that generalize and/or improve previous results regarding these questions and help to better understand them. In complexity theory, a sparse set is a set with polynomially bounded density. Whether sparse sets are hard for complexity classes is one of the central problems in complexity theory [5,12,13,17]. In particular, Mahaney [13] showed that sparse sets cannot be many-one complete for NP unless $P=NP$. In Section 3 we study the question whether sparse sets can be hard for NE under reductions that are weaker than the polynomial-time Turing reductions. We prove that no NE-hard set can be reducible to sparse sets via the *strong nondeterministic polynomial-time many-one reduction*. For a special case of sparse sets, tally sets,

we strengthen the result to the *nondeterministic polynomial-time Turing reductions* that make at most $n^{o(1)}$ many queries. These are the main results of this paper. Note that generalizing these results to polynomial-time Turing reductions is hard since already the deterministic polynomial-time Turing reduction closure of sparse sets as well as that of p-selective sets equals $P/poly$ [10], and it is not even known whether $NE \not\subseteq P/poly$.

We present a new result on the aforementioned long standing open question whether every NE set is polynomial-time Turing-reducible to a NP set. Fu et al. [7] first tackled this problem and showed that $NE \not\subseteq P_{n^{o(1)}-T}(NP)$. Their result was later improved by Mocas [14] to $NEXP \not\subseteq P_{n^c-T}(NP)$ for any constant $c > 0$. Mocas's result is optimal with respect to relativizable proofs, as Buhrman and Torenvliet [3] constructed an oracle relative to which $NEXP = P^{NP}$. In this paper, we extend Mocas's result to nonuniform polynomial-time Turing reductions that uses a fixed polynomial number of advice bits. More precisely, we show that $NE \not\subseteq P_{n^k-T}(NP)/n^{k'}$ for any constant $k, k' > 0$. Since it is easy to show for any $k > 0$ that $P_{n^k-T}(NP \oplus P\text{-Sel}) \subseteq P_{n^k-T}(NP)/n^k$, where P-Sel denotes the class of p-selective sets, we obtain as a corollary that $NE \not\subseteq P_{n^k-T}(NP \oplus P\text{-Sel})$.

We investigate a different but related question. We study the question of how different a hard problem in NE is from a problem in NP. One way to measure the difference between sets is by using the notion of *closeness* introduced by Yesha [19]. We say two sets are f -close if the density of their symmetric difference is bounded by $f(n)$. The closeness to NP-hard sets were further studied by Fu [6] and Ogihara [16]. We show that for every $\leq_m^P$ -complete set H for NE and every NP-set $A \subseteq H$, there exists another NP-set $A' \subseteq H$ such that $A \subseteq A'$ and A' is not subexponential-close to A . For coNE-complete sets we show a stronger result. We show that for every $\leq_m^P$ -complete set H for coNE and every NP-set $A \subseteq H$, there exists another NP-set $A' \subseteq H$ such that $A \cap A' = \emptyset$ and A' is exponentially dense.

2 Notations

We use standard notations [11,9] in structural complexity. All the languages throughout the paper are over the alphabet $\Sigma = \{0, 1\}$. For a string x , $|x|$ is the length of x . For a finite set A , $|A|$ is the number of elements in A . We use Σ^n to denote the set of all strings of length n and for any language L , $L^n = L \cap \Sigma^n$. We fix a pairing function $\langle \cdot \rangle$ such that for every $u, v \in \Sigma^*$, $|\langle u, v \rangle| = 2(|u| + |v|)$. For a function $f(n) : N \rightarrow N$, f is *exponential* if for some constant $c > 0$, $f(n) \geq 2^{n^c}$ for all large n , and is *sub-exponential* if for every constant $c > 0$, $f(n) \leq 2^{n^c}$ for all large n . A language L is *exponentially dense* if there exists a constant $c > 0$ such that $||L^{\leq n}|| \geq 2^{n^c}$ for all large n . Let $\text{Density}(d(n))$ be the class of languages L such that $||L^{\leq n}|| \leq d(n)$ for all large n . For any language L , define its *complementary language*, denoted by $\overline{L}$, to be $\Sigma^* - L$.

For a function $t(n) : N \rightarrow N$, $\text{DTIME}(t(n))$ ($\text{NTIME}(t(n))$) is the class of languages accepted by (non)-deterministic Turing machines in time $t(n)$. P (NP) is

the class of languages accepted by (non-)deterministic polynomial-time Turing machines. E (NE) is the class of languages accepted by (non-)deterministic Turing machines in $2^{O(n)}$ time. EXP (NEXP) is the class of languages accepted by (non-)deterministic Turing machines in time $2^{n^{O(1)}}$. TALLY is the class of languages contained in 1^* and SPARSE is the class of languages in $\cup_{c=1}^{\infty} \text{Density}(n^c)$. Clearly, TALLY is a subclass of SPARSE. We use P-Sel to denote the class of p-selective sets [18]. For any language L and function $h : N \mapsto N$, let $L/h = \{x : \langle x, h(|x|) \rangle \in L\}$. For any class $\mathcal{C}$ of languages, $\text{co}\mathcal{C}$ is the class of languages L such that $\bar{L} \in \mathcal{C}$ and $\mathcal{C}/h$ is the class of languages L such that $L = L'/h$ for some $L' \in \mathcal{C}$.

For two languages A and B , define the following reductions: (1) A is *polynomial-time many-one reducible* to B , $A \leq_m^p B$, if there exists a polynomial-time computable function $f : \Sigma^* \mapsto \Sigma^*$ such that for every $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$. (2) A is *polynomial-time truth-table reducible* to B , $A \leq_{tt}^p B$, if there exists a polynomial-time computable function $f : \Sigma^* \mapsto \Sigma^*$ such that for every $x \in \Sigma^*$, $f(x) = \langle y_1, y_2, \dots, y_m, T \rangle$, where $y_i \in \Sigma^*$ and T is the encoding of a circuit, and $x \in A$ if and only if $T(B(y_1)B(y_2) \dots B(y_m)) = 1$. (3) A is *polynomial-time Turing reducible* to B , $A \leq_T^p B$, if there exists a polynomial-time oracle Turing machine M such that M^B accepts A . (4) A is *exponential-time Turing reducible* to B , $A \leq_T^{\text{EXP}} B$, if there exists an exponential-time oracle Turing machine M such that M^B accepts A . (5) We say $A \leq_1^p B$ if $A \leq_m^p B$ via a reduction f that is one-to-one.

For a nondeterministic Turing machine M , denote $M(x)[y]$ to be the computation of M with input x on a path y . If $M(x)$ is an oracle Turing machine, $M^A(x)[y]$ is the computation of M with input x on a path y with oracle A .

For two languages A and B , define the following nondeterministic reductions: (1) A is *nondeterministically polynomial-time many-one reducible* to B , $A \leq_m^{\text{NP}} B$, if there exists a polynomial-time nondeterministic Turing machine M and a polynomial $p(n)$ such that for every x , $x \in A$ if and only if there exists a path y of length $p(|x|)$ with $M(x)[y] \in B$. (2) A is *nondeterministically polynomial-time truth-table reducible* to B , $A \leq_{tt}^{\text{NP}} B$, if there exists a polynomial-time nondeterministic Turing machine M and a polynomial $p(n)$ such that for every $x \in \Sigma^*$, $x \in A$ if and only if there is at least one $y \in \Sigma^{p(|x|)}$ such that $M(x)[y] = (z_1, \dots, z_m, T)$, where $z_i \in \Sigma^*$, T is the encoding of a circuit, and $T(B(z_1), \dots, B(z_m)) = 1$. (3) A is *nondeterministically polynomial-time Turing reducible* to B , $A \leq_T^{\text{NP}} B$, if there exists a polynomial-time nondeterministic oracle Turing machine M and a polynomial p such that for every $x \in \Sigma^*$, $x \in A$ if and only if there is at least one $y \in \Sigma^{p(|x|)}$ such that $M^B(x)[y]$ accepts. (4) A is *strongly nondeterministically polynomial-time many-one reducible* to B , $A \leq_m^{SN} B$, if there exists a polynomial-time nondeterministic Turing machine $M()$ such that $x \in A$ if and only if 1) $M(x)[y] \in B$ for all y that $M(x)[y]$ is not empty; 2) $M(x)[y]$ is not empty for at least one $y \in \Sigma^{n^{O(1)}}$.

For a function $g(n) : N \rightarrow N$, we use $A \leq_{g(n)-tt}^{\text{NP}} B$ to denote that $A \leq_{tt}^{\text{NP}} B$ via a polynomial-time computable function f such that for every $x \in \Sigma^n$, $f(x, y) = (z_1, \dots, z_m, T)$ and $m \leq g(n)$. We use $A \leq_{btt}^{\text{NP}} B$ to denote that $A \leq_{c-tt}^{\text{NP}} B$.

B for some constant $c > 0$. For $t \in \{p, \text{NP}, \text{EXP}\}$, we use $A \leq_{g(n)-T}^t$ to denote that $A \leq_T^t$ via a Turing machine M that makes at most $g(n)$ queries on inputs of length n .

For a class $\mathcal{C}$ of languages, we use $R_r^t(\mathcal{C})$ ($R_{g(n)-r}^t(\mathcal{C})$) to denote the reduction closure of $\mathcal{C}$ under the reduction $\leq_r^t$ ($\leq_{g(n)-r}^t$), where $r \in \{p, \text{NP}, \text{SN}, \text{EXP}\}$ and $r \in \{m, tt, T\}$. We also use conventional notations for common reduction closures such as $\text{P}^{\text{NP}} = \text{P}_T(\text{NP}) = R_T^p(\text{NP})$ and $\text{EXP}_{n^k-T}^{\text{NP}} = \text{EXP}_{n^k-T}(\text{NP}) = R_{n^k-T}^{\text{EXP}}(\text{NP})$. For a function $l : N \mapsto N$ and a reduction closure R , we use $R[l(n)]$ to denote the same reduction closure as R except that the reductions make queries of length at most $l(n)$ on inputs of length n .

A function $f(n)$ from N to N is time constructible if there exists a Turing machine M such that $M(n)$ outputs $f(n)$ in $f(n)$ steps.

3 Separating NE from $R_{n^{o(1)}-T}^{\text{NP}}(\text{TALLY})$

In this section, we present the main result that NE cannot be reduced to TALLY via polynomial time Turing reduction with the number of queries bounded by $n^{1/\alpha(n)}$ for some polynomial time computable nondecreasing function $\alpha(n)$ (for example, $\alpha(n) = \log \log n$). The proof is a combination of the translational method and the point of view from Kolmogorov complexity.

Lemma 1. *Assume that function $g(n) : N \rightarrow N$ is nondecreasing unbounded and function $2^{\lfloor n \rfloor^{g(n)}/2}$ is time constructible. Then there exists a language $L_0 \in \text{DTIME}(2^{n^{g(n)}})$ such that $\|L_0^n\| = 1$, and for every Turing machine M , M cannot generate any sequence in L_0^n with any input of length $n - \log n$ in $2^{n^{o(1)}}$ time for large n .*

Proof. We use the diagonal method to construct the language L_0 . Let $M_1, \dots, M_k, \dots$ be an enumeration of all Turing transducers.

Construction:

Input n ,

Simulate each machine $M_i(y)$ in $2^{n^{g(n)}/2}$ steps for $i = 1, \dots, \log n$ and all y of length $n - \log n$.

Find a string x of length n such that x cannot be generated by any machine among $M_1, \dots, M_{\log n}$ with any input of length at most $n - \log n$.

Put x into L_0 .

End of Construction

There are at most $2^{n - \log n + 1}$ strings of length at most $n - \log n$. Those $\log n$ machines can generate at most $2^{n - \log n + 1} \log n < 2^n$ strings. Since generating each string takes $2^{n^{g(n)}/2}$ steps. This takes $2^n \cdot 2^{n^{g(n)}/2} < 2^{n^{g(n)}}$ time for all large n . $\square$

Theorem 1. *Assume that $t(n)$ and $f(n)$ are time constructible nondecreasing functions from N to N such that 1) $t(f(n))$ is $\Omega(2^{n^{g(n)}})$ for some nondecreasing*

unbounded function $g(n)$, and 2) for any constant $c > 0$, $f(n) \leq t(n)^{1/c}$ and $f(n) \geq 4n$ for all large n . If $q(n)$ is a nondecreasing function with $q(f(n))(\log f(n)) = o(n)$, then $\text{NTIME}(t(n)) \not\subseteq R_{q(n)-T}^{\text{NP}}(\text{TALLY})$.

Proof. We apply a translational method to obtain such a separation. We prove by contradiction and assume that $\text{NTIME}(t(n)) \subseteq R_{q(n)-T}^{\text{NP}}(\text{TALLY})$. Without loss of generality, we assume that $q(n) \geq 1$.

Let L be an arbitrary language in $\text{DTIME}(t(f(n)))$. Define $L_1 = \{x10^{f(|x|)-|x|-1} : x \in L\}$. It is easy to see that L_1 is in $\text{DTIME}(t(n))$ since L is in $\text{DTIME}(t(f(n)))$.

By our hypothesis, there exist a set $A_1 \in \text{TALLY}$ such that $L_1 \leq_{q(n)-T}^{\text{NP}} A_1$ via some polynomial time nondeterministic oracle Turing machine M_1 , which runs in polynomial n^{c_1} time for all large n .

Let $L_2 = \{(x, (e_1, \dots, e_m, a_1 \dots a_m)) : \text{there is a path } y \text{ such that } M_1^{A_1}(x10^{f(|x|)-|x|-1})[y] \text{ accepts and queries } 1^{e_1}, \dots, 1^{e_m} \text{ in path } y \text{ and receives answers } a_1 = A_1[1^{e_1}], \dots, a_m = A_1[1^{e_m}] \text{ respectively}\}$. Since M_1 runs in time n^{c_1} and $f(n) = t(n)^{o(1)}$, we have L_2 is in $\text{NTIME}(f(n)^{c_1}) \subseteq \text{NTIME}(t(n))$.

By our hypothesis, there exists a set $A_2 \in \text{TALLY}$ such that $L_2 \leq_{q(n)-T}^{\text{NP}} A_2$ via some polynomial time nondeterministic oracle Turing machine $M_2(\cdot)$.

Therefore, for every string x , in order to generate $x \in L$, we need to provide $(e_1, \dots, e_m, a_1 \dots a_m)$ and $(z_1, \dots, z_t, b_1 \dots b_t)$ such that there exists a path y_1 that $M_1^{A_1}(x10^{f(|x|)-|x|-1})[y_1]$ queries $1^{e_1}, \dots, 1^{e_m}$ with answers $a_i = A_1(1^{e_i})$ for $i = 1, \dots, m$ and there exists a path y_2 that $M_2^{A_2}(x, (e_1, \dots, e_m, a_1 \dots a_m))[y_2]$ queries $1^{z_1}, \dots, 1^{z_t}$ with $b_i = A_2(1^{z_i})$ for $i = 1, \dots, t$. Let n^{c_2} be the polynomial time bound for M_2 . We have the following Turing machine M^* .

$M^*(\cdot)$:

Input: a string of u of length $o(n)$.

If u does not have the format $(e_1, \dots, e_m, a_1 \dots a_m)(z_1, \dots, z_t, b_1 \dots b_t)$, then return λ (empty string).

Extract $(e_1, \dots, e_m, a_1 \dots a_m)$ and $(z_1, \dots, z_t, b_1 \dots b_t)$ from u .

For each x of length n

Simulate $M_2^{A_2}(x, (e_1, \dots, e_m, a_1 \dots a_m))$ with the query help from $(z_1, \dots, z_t, b_1 \dots b_t)$ (by assuming that $b_i = A_2(1^{z_i})$ for $i = 1, \dots, t$).

Output x if it accepts.

It is easy to see that M^* takes $2^{n^{O(1)}}$ time. There exists a path y_1 such that $M_1^{A_1}(x10^{f(|x|)-|x|-1})[y_1]$ makes at most $q(f(n))$ queries, where $n = |x|$. So, we have $m \leq q(f(n))$, $e_i \leq f(n)^{c_1}$ and $|e_i| \leq c_1(\log f(n))$. Therefore, $(e_1, \dots, e_m, a_1, \dots, a_m)$ has length $h \leq 2(O(q(f(n)) \log f(n)) + q(f(n))) = O(q(f(n)) \log f(n)) = o(n)$. There exists a path y_2 such that $M_2^{A_2}((x, (e_1, \dots, e_m, a_1 \dots a_m))[y_2]$ makes at most $q(n+h)$ queries to $1^{z_1}, \dots, 1^{z_t}$. The length of $(x, (e_1, \dots, e_m, a_1 \dots a_m))$ is at most $2(n+h) \leq 4n$. So, $t \leq q(4n)$. Therefore, $(z_1, \dots, z_t, b_1 \dots b_t)$ has length $q(4n) \log((4n)^{c_2}) = O(q(f(n)) \log f(n)) = o(n)$. Therefore, the total length of $(e_1, \dots, e_m, a_1 \dots a_m)$ and $(z_1, \dots, z_t, b_1 \dots b_t)$ is $o(n)$. So, $(e_1, \dots, e_m, a_1 \dots a_m)$ and $(z_1, \dots, z_t, b_1 \dots b_t)$ can be encoded into a

string of length $o(n)$. Let L be the language L_0 in Lemma 1. This contradicts Lemma 1 since a string of length n can be generated by M^* with the input $(e_1, \dots, e_m, a_1 \dots a_m)(z_1, \dots, z_t, b_1 \dots b_t)$ of length $o(n)$. $\square$

Corollary 1. $\text{NE} \not\subseteq R_{n^{1/\alpha(n)}-T}^{\text{NP}}(\text{TALLY})$ for any polynomial computable non-decreasing unbounded function $\alpha(n) : N \rightarrow N$.

Proof. Define $g(n) = \lfloor \sqrt{\alpha(n)} \rfloor$, $f(n) = n^{g(n)}$, $q(n) = n^{\frac{1}{\alpha(n)}}$, and $t(n) = 2^n$. By Theorem 1, we have that $\text{NTIME}(t(n)) \not\subseteq R_{q(n)-T}^{\text{NP}}(\text{TALLY})$. We have that $\text{NE} \not\subseteq R_{n^{1/\alpha(n)}-T}^{\text{NP}}(\text{TALLY})$ since $R_{n^{1/\alpha(n)}-T}^{\text{NP}}(\text{TALLY})$ is closed under $\leq_m^P$ reductions and there exists a NE - $\leq_m^P$ -hard set in $\text{NTIME}(t(n))$. $\square$

It is natural to extend Theorem 1 by replacing TALLY by SPARSE . We feel it is still hard to separate NE from $R_m^{\text{NP}}(\text{SPARSE})$. The following theorem shows that we can separate NE from $R_m^{\text{SN}}(\text{SPARSE})$. Its proof is another application of the combination of translational method with Kolmogorov complexity point of view.

Theorem 2. Assume that $t_0(n)$ and $t(n)$ are time constructible nondecreasing functions from N to N such that for any positive constant c , $t_0(n)^c = O(t(n))$ and $t(t_0(n)) > 2^{n^{\alpha(n)}}$ for some nondecreasing unbounded function $\alpha(n)$, and $d(n)$ is a nondecreasing function such that $d((t_0(n))^c) = 2^{n^{c^{(1)}}}$. Then $\text{NTIME}(t(n)) \not\subseteq R_m^{\text{SN}}(\text{Density}((d(n))))$.

Proof. Assume that $\text{NTIME}(t(n)) \subseteq R_m^{\text{SN}}(\text{Density}(d(n)))$. We will derive a contradiction.

Construction of L^n : Let S be the sequence of length $n^{1+\frac{1}{k}}$ in L_0 of Lemma 1 with $g(n) = \alpha(n)$, where $n = m^k$ and k is a constant (for example $k = 100$). Assume that $S = y_1 y_2 \dots y_{m^2}$, where each y_i is of length m^{k-1} . Let $L^n = \{y_{i_1} y_{i_2} \dots y_{i_m} : 1 \leq i_1 < i_2 < \dots < i_m \leq m^2\}$. Define $\text{block}(x) = \{y_{i_1}, y_{i_2}, \dots, y_{i_m}\}$ if $x = y_{i_1} y_{i_2} \dots y_{i_m}$. Clearly, L^n contains $\binom{m^2}{m}$ elements.

Define $L_1 = \{x 10^{t_0(|x|)-|x|-1} : x \in L\}$. It is easy to see that L_1 is in $\text{DTIME}(t(n))$ since L is in $\text{DTIME}(t(t_0(n)))$.

By our hypothesis, there exists a set $A_1 \in \text{Density}(d(n))$ such that $L_1 \leq_m^{\text{SN}} A_1$ via some polynomial time nondeterministic Turing machine $f()$, which runs in polynomial time n^{c_1} . For a sequence z and integer n , define $H(z, n) = \{x \in L^n : f(x)[y] = z \text{ for some path } y\}$. Therefore, there are a sequence z such that

$$||H(z, n)|| \geq \frac{\binom{m^2}{m}}{d((t_0(n))^{c_1})}.$$

Let $L_2 = \{(x, y) : |x| = |y| \text{ and there are paths } z_1 \text{ and } z_2 \text{ such that } f(x 10^{t_0(|x|)-|x|-1})[z_1] = f(y 10^{t_0(|y|)-|y|-1})[z_2]\}$. Since $f()$ runs in polynomial time and $t_0(n)^{c_1} = O(t(n))$, we have $L_2 \in \text{NTIME}(t(n))$.

By our hypothesis, there exists a set $A_2 \in \text{Density}(d(n))$ with such that $L_2 \leq_m^{\text{SN}} A_2$ via some polynomial time nondeterministic Turing machine $u()$.

Define $L_2(x) = \{x_1 : (x, x_1) \in L_2\}$. There exists $x \in L^n$ such that $||L_2(x)|| \geq \frac{\binom{m^2}{m}}{d((t_0(n))^{c_1})}$.

Define $L'_2(x, x') = \{x_2 : u(x, x')[z'] = u(x, x_2)[z_2] \text{ for some paths } z' \text{ for } u(x, x') \text{ and } z_2 \text{ for } u(x, x_2)\}$. There exists $x' \in L_2(x)$ such that $L'_2(x, x')$ contains at least $\frac{\binom{m^2}{m}}{d((t_0(n))^{c_1})d((t_0(n))^{c_2})}$ elements. We fix x and x' .

Since $|\text{block}(x) \cup \text{block}(x')| \leq 2m$, those $2m$ strings in $\text{block}(x) \cup \text{block}(x')$ can generate at most $\binom{2m}{m} < \frac{\binom{m^2}{m}}{d((t_0(n))^{c_1})d((t_0(n))^{c_2})}$ sequences of length n in L^n for all large n . Therefore, there is a string $x_3 \in L^n$ such that $x_3 \in L'_2(x, x')$ and $\text{block}(x_3) \not\subseteq \text{block}(x) \cup \text{block}(x')$.

This makes it possible to compress S . We can encode the strings x, x' and those blocks of S not in x_3 . The total time is at most $2^{n^{O(1)}}$ to compress S .

Let $y_{i_1} < y_{i_2} < \dots < y_{i_{m^2}}$ be the sorted list of $y_1, y_2, \dots, y_{m^2}$. Let $(i_1, i_2, \dots, i_{m^2})$ be encoded into a string of length $O(m^2(\log n))$. Define $Y = y_{j_1}y_{j_2} \dots y_{j_t}$, where $\{y_{j_1}, y_{j_2}, \dots, y_{j_t}\} = \{y_1, \dots, y_{m^2}\} - (\text{block}(x) \cup \text{block}(x') \cup \text{block}(x_3))$.

We can encode $(i_1, i_2, \dots, i_{m^2})$ into the format $0a_10a_2 \dots 0a_u11$. We have sequence $Z = (i_1, i_2, \dots, i_{m^2})xx'Y$ to generate S in $2^{n^{O(1)}}$ time. Since at least one block y_i among $y_1, y_2, \dots, y_{m^2}$ is missed in $\text{block}(xx'Y)$, $|y_i| = m^{k-1}$, and $|(i_1, i_2, \dots, i_{m^2})| < m^3$, it is easy to see that $|Z| \leq n - (\log n)^2$. This brings a contradiction. $\square$

Corollary 2. $\text{NE} \not\subseteq R_m^{SN}(\text{SPARSE})$.

Proof. Let $t(n) = 2^n$, $t_0(n) = n^{\log n}$, and $d(n) = n^{\log n}$. Apply Theorem 2. $\square$

4 On the Differences between NE and NP

In this section we investigate the differences between NE-hard sets and NP sets. We use the following well-known result:

Lemma 2 ([8]). *Let H be $\leq_m^p$ -hard for NE and $A \in \text{NE}$. Then $A \leq_1^p H$.*

Theorem 3. *For every set H and $A \subseteq H$ such that H is $\leq_m^p$ -hard for NE and $A \in \text{NP}$, there exists another set $A' \subseteq H$ such that $A' \in \text{NP}$ and $A' - A$ is not of subexponential density.*

Proof. Fix H and A as in the premise and let $A \in \text{NTIME}(n^c)$ for some constant $c > 1$. Let $\{NP_i\}_i$ be an enumeration of all nondeterministic polynomial-time Turing machines such that the computation NP_i on x can be simulated nondeterministically in time $2^{O((|i| + \log(|x|))^2)}$ [8]. Define $S = \{\langle i, x, y \rangle : x, y \in \Sigma^* \text{ and } NP_i \text{ accepts } x\}$. Clearly S belongs to NEXP and therefore S is many-one reducible to H via some polynomial-time computable one-one function f . Suppose f can be computed in time n^d for some $d > 1$. By Cook [4], let $B \in \text{NP} - \text{NTIME}(n^{2cd})$. Suppose $B = L(NP_i)$ for some i . For each $x \in \Sigma^*$, define $T_x = \{z : \exists y(|x| = |y|/2 \leq |z| \text{ and } z = f(\langle i, x, y \rangle))\}$. Let $T = \bigcup_{x \in B} T_x$. Clearly $T \in \text{NP}$. Since f reduces S to H , $T_x \subseteq H$ for all $x \in B$ and therefore $T \subseteq H$. We now establish the following claims:

Claim 1. For infinitely many $x \in B$, $A \cap T_x = \emptyset$.

Proof. Suppose not. Consider the following machine M :

- 0 On input x
- 1 Guess y with $|y| = 2|x|$;
- 2 Compute $z = f(\langle i, x, y \rangle)$;
- 3 Accept x if and only if $|z| \geq |x|$ and $z \in A$.

Assume $x \in B$ and $A \cap T_x \neq \emptyset$. Let $z \in A \cap T_x$ and hence there exists y with $|y|/2 = |x| \leq |z|$ and $z = f(\langle i, x, y \rangle)$. Thus, M accepts x if it correctly guess y in line 1. Now assume $x \notin B$. Then $T_x \subseteq \overline{H}$ and hence $A \cap T_x = \emptyset$. Thus, for any z computed in line 3, $z \notin A$. So M does not accept x . This shows that M decides B for all but finitely many x . However, the machine M runs in time $O(((2|x|)^d)^c) = O((|x|)^{cd})$ for sufficiently large x , which contradicts that $B \notin \text{NTIME}(n^{2cd})$. $\square$

Claim 2. For any infinite set R , the set $\cup_{x \in R} T_x$ is not in $\text{Density}(f(n))$ for any sub-exponential function $f : N \rightarrow N$.

Proof. Let R be an infinite set and $T' = \cup_{x \in R} T_x$. Fix a string x . Since f is a one-one function, $\|\{f(\langle i, x, y \rangle)\}_{|y|=2|x|}\| = 2^{2|x|}$. Since there are only $2^{|x|}$ of strings of length less than $|x|$, it follows that there are at least $2^{2|x|} - 2^{|x|} \geq 2^{|x|}$ many strings in T_x . Note that the strings in T_x have lengths at most $\Theta(|x|^d)$ and hence, $\|(T')^{\leq \Theta(|x|^d)}\| \geq 2^{|x|}$. Since x is arbitrary, this shows that $\cup_{x \in R} T_x$ is not $\text{Density}(f(n))$ for any sub-exponential function $f : N \rightarrow N$. $\square$

Now Let $A' = A \cup T$. By Claims 1 and 2, A' clearly has all the desired properties. $\square$

Theorem 3 shows that many-one-hard sets for NE are very different from their NP subsets. Namely they're not even sub-exponentially close to their NP subsets. Next we show a stronger result for many-one-hard sets for coNE. We show that the difference between a many-one-hard set for coNE and any of its NP subset has exponential density.

Theorem 4. Assume that H is a many-one-hard set for coNE and $t(n) : N \rightarrow N$ is a sub-exponential function. Then for any $A \subseteq H$ with $A \in \text{NTIME}(t(n))$, there exists another set $A' \subseteq H$ such that $A' \in \text{NP}$, $A' \cap A = \emptyset$, and A' is exponentially dense.

Proof. Fix H and A as in the premise. By a result of Fu et al. [7, Corollary 4.2], $H' = \overline{H} \cup A$ is many-one hard for NE. Now let f be a polynomial-time one-one reduction from $0\Sigma^*$ to H' and suppose f is computable in time n^d . Let $A' = \{z : z = f(1x) \text{ for some } x \text{ with } |x| \leq 2|z|\}$. Clearly $A' \in \text{NP}$ and $A' \subseteq \overline{H'}$. Therefore $A' \subseteq H - A$. It remains to show that A' is exponentially dense. For any $n > 0$, let $F_n = \{f(1x)\}_{|x|=2n}$. Since f is one-one, $\|F_n\| = 2^{2n}$. As there are only 2^n strings of length less than n , it follows that there are at least $2^{2n} - 2^n \geq 2^n$ many strings in F_n belonging to A' for each $n > 0$. Note that the maximal length of a string in F_n is $(2n+1)^d$. This shows that $(A')^{\leq (2n+1)^d} \geq 2^n$ for each $n > 0$ and hence, A' is exponentially dense. $\square$

Corollary 3. *Assume that H is a $\leq_m^P$ -hard set for coNE. Then for $A \subseteq H$ with $A \in \text{NP}$, there exists another subset $A' \subseteq H$ such that $A' \in \text{NP}$, $A' \cap A = \emptyset$, and A' is exponentially dense.*

5 Separating NE from $\text{P}_{n^k-T}^{\text{NP}}$ for Nonuniform Reductions

In this section we generalize Mocas's result [14] that $\text{NEXP} \not\subseteq \text{P}_{n^c-T}(\text{NP})$ for any constant $c > 0$ to non-uniform Turing reductions.

Lemma 3. *For any positive constants $k, k' > 0$, $\text{EXP}_{n^k-T}^{\text{NP}} \not\subseteq \text{P}_{n^k-T}^{\text{NP}}/n^{k'}$.*

Proof. Burtschick and Linder [2] showed that $\text{DTIME}(2^{4f(n)}) \not\subseteq \text{DTIME}(2^{f(n)})/f(n)$ for any function $f : N \rightarrow N$ with $n \leq f(n) < 2^n$. Applying their result with $f(n) = n^{k'}$ yields $\text{EXP} \not\subseteq \text{P}/n^{k'}$ for any $k' > 0$. The lemma follows by noting the fact that Burtschick and Linder's result also holds relative to any oracle. $\square$

Theorem 5. *For any positive constants $k, k' > 0$, $\text{NEXP} \not\subseteq \text{P}_{n^k-T}^{\text{NP}}/n^{k'}$.*

Proof. Assume that $\text{NEXP} \subseteq \text{P}_{n^k-T}^{\text{NP}}/n^{k'}$ for some $k, k' > 0$. Since $\text{EXP}_{n^k-T}^{\text{NP}} \subseteq \text{P}_T^{\text{NEXP}}[n^{k+1}]$ [14], we have $\text{EXP}_{n^k-T}^{\text{NP}} \subseteq \text{P}_T(\text{P}_{n^k-T}^{\text{NP}}/n^{k'})[n^{k+1}] \subseteq \text{P}_T^{\text{NP}}/(n^{k+1})^{k'} \subseteq \text{NEXP}/n^{(k+1)k'} \subseteq (\text{P}_{n^k-T}^{\text{NP}}/n^{k'})/n^{(k+1)k'} \subseteq \text{P}_{n^k-T}^{\text{NP}}/n^{k''}$ for some $k'' > 0$. The last inclusion is a contradiction to Lemma 3. $\square$

Since any NEXP set can be easily padded to an NE set, we immediately obtain the following corollary:

Corollary 4. *For any positive constants $k, k' > 0$, $\text{NE} \not\subseteq \text{P}_{n^k-T}^{\text{NP}}/n^{k'}$.*

Lemma 4. *For any $k > 0$, $\text{P}_{n^k-T}(\text{NP} \oplus \text{P-Sel}) \subseteq \text{P}_{n^k-T}(\text{NP})/n^k$.*

Proof. Assume that $L \in \text{P}_{n^k-T}(\text{NP} \oplus \text{P-Sel})$ via polynomial time Turing reduction D . Let A be a P-selective set with order $\preceq$ such that A is an initial segment with $\preceq$ and $L \in \text{P}_{n^k-T}(\text{SAT} \oplus A)$ via D . Let y be the largest element in A (with the order $\preceq$) queried by $D^{\text{SAT} \oplus A}$ among all inputs of length $\leq n$. It is easy to see that y can be generated by simulating D with advice of length n^k . When we compute $D^{\text{SAT} \oplus A}(x)$, we handle the queries to A by comparing with y . $\square$

By Theorem 5 and Lemma 4, we have the following theorem.

Theorem 6. *For any constant $k > 0$, $\text{NE} \not\subseteq \text{P}_{n^k-T}(\text{NP} \oplus \text{P-Sel})$.*

6 Conclusions

We derived some separations between NE and other nondeterministic complexity classes. The further research along this line may be in separating NE from P_T^{NP} , and NE from BPP, which is a subclass of P/Poly.

Acknowledgements. We thank unknown referees for their helpful comments. Bin Fu is supported in part by National Science Foundation Early Career Award 0845376.

References

1. Berman, L., Hartmanis, J.: On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing* 6(2), 305–322 (1977)
2. Bertschick, H.-J., Lindner, W.: On sets Turing reducible to p-selective sets. *Theory of Computing Systems* 30, 135–143 (1997)
3. Buhrman, H., Torenvliet, L.: On the Cutting Edge of Relativization: The Resource Bounded Injury Method. In: Shamir, E., Abiteboul, S. (eds.) *ICALP 1994*. LNCS, vol. 820, pp. 263–273. Springer, Heidelberg (1994)
4. Cook, S.: A Hierarchy for Nondeterministic Time Complexity. *J. Comput. Syst. Sci.* 7(4), 343–353 (1973)
5. Cai, J., Sivakumar, D.: Sparse hard sets for P: resolution of a conjecture of hartmanis. *Journal of Computer and System Sciences* (0022-0000) 58(2), 280–296 (1999)
6. Fu, B.: On lower bounds of the closeness between complexity classes. *Mathematical Systems Theory* 26(2), 187–202 (1993)
7. Fu, B., Li, H., Zhong, Y.: Some properties of exponential time complexity classes. In: *Proceedings 7th IEEE Annual Conference on Structure in Complexity Theory*, pp. 50–57 (1992)
8. Ganesan, K., Homer, S.: Complete Problems and Strong Polynomial Reducibilities. *SIAM J. Comput.* 21(4), 733–742 (1992)
9. Hemaspaandra, L., Ogihara, M.: *The Complexity Theory Companion*. Texts in Theoretical Computer Science - An EATCS Series. Springer, Heidelberg (2002)
10. Hemaspaandra, L., Torenvliet, L.: *Theory of Semi-Feasible Algorithms*. Springer, Heidelberg (2003)
11. Homer, S., Selman, A.: Computability and Complexity Theory. In: *Texts in Computer Science*, Springer, New York (2001)
12. Karp, R., Lipton, R.: Some connections between nonuniform and uniform complexity classes. In: *Proceedings of the twelfth annual ACM symposium on theory of computing*, pp. 302–309 (1980)
13. Mahaney, S.: Sparse complete sets for NP: Solution of a conjecture of berman and hartmanis. *Journal of Computer and Systems Sciences* 25(2), 130–143 (1982)
14. Mocas, S.: Separating classes in the exponential-time hierarchy from classes in PH. *Theoretical Computer Science* 158, 221–231 (1996)
15. Ogihara, M., Tantau, T.: On the reducibility of sets inside NP to sets with low information content. *Journal of Computer and System Sciences* 69, 499–524 (2004)
16. Ogiwara, M.: On P-closeness of polynomial-time hard sets (unpublished manuscript, 1991)
17. Ogiwara, M., Watanabe, O.: On polynomial-time bounded truth-table reducibility of NP sets to sparse sets. *SIAM Journal on Computing* 20(3), 471–483 (1991)
18. Selman, A.: P-selective sets, tally languages and the behavior of polynomial time reducebilities on NP. *Mathematical Systems Theory* 13, 55–65 (1979)
19. Yesha, Y.: On certain polynomial-time truth-table reducibilities of complete sets to sparse sets. *SIAM Journal on Computing* 12(3), 411–425 (1983)

On the Readability of Monotone Boolean Formulae

Khaled Elbassioni¹, Kazuhisa Makino², and Imran Rauf¹

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany
`{elbassio, irauf}@mpi-inf.mpg.de`

² Department of Mathematical Informatics, University of Tokyo, Tokyo, Japan
`makino@mist.i.u-tokyo.ac.jp`

Abstract. Golumbic et al. [Discrete Applied Mathematics 154(2006) 1465-1477] defined the *readability* of a monotone Boolean function f to be the minimum integer k such that there exists an $\wedge - \vee$ -formula equivalent to f in which each variable appears at most k times. They asked whether there exists a polynomial-time algorithm, which given a monotone Boolean function f , in CNF or DNF form, checks whether f is a read- k function, for a fixed k . In this paper, we partially answer this question already for $k = 2$ by showing that it is NP-hard to decide if a given monotone formula represents a read-twice function. It follows also from our reduction that it is NP-hard to approximate the readability of a given monotone Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ within a factor of $\mathcal{O}(n)$. We also give tight sublinear upper bounds on the readability of a monotone Boolean function given in CNF (or DNF) form, parameterized by the number of terms in the CNF and the maximum size in each term, or more generally the maximum number of variables in the intersection of any constant number of terms. When the variables of the DNF can be ordered so that each term consists of a set of consecutive variables, we give much tighter polylogarithmic bounds on the readability.

1 Introduction

Let $f : \{0, 1\}^n \rightarrow \{0, 1\}$ be a monotone Boolean function, i. e., for any $x, x' \in \{0, 1\}^n$, $x' \geq x$ implies $f(x') \geq f(x)$. One property of such functions is that they can be represented by negation-free Boolean formulae. A *minterm* (*maxterm*) of monotone Boolean function $f(x_1, \dots, x_n)$ is a minimal set of variables which, if assigned the value 1 (resp., value 0), forces the function to take the value 1 (resp., value 0) regardless of the values assigned to the remaining variables. It is well-known that the irredundant (i. e., no term contains another) *disjunctive normal form* (DNF) and *conjunctive normal form* (CNF) of monotone Boolean function f consist respectively of all of its minterms and maxterms (cf. [Weg87]).

A monotone read- k formula is a Boolean formula over the operators $\{\vee, \wedge\}$ in which each variable occurs at most k times. The readability of f is the minimum k such that f can be represented by a monotone read- k formula. We also call f a read- k function when it has readability k . Finding the readability of an arbitrary

Boolean function and computing a formula which achieves this readability has applications in circuit design among others and therefore is one of the earliest problems considered in Computer Science [GMR06].

Given a monotone Boolean function in one of the normal forms (CNF/DNF), a complete combinatorial characterization for it to be read-once was given by Gurvich [Gur77]. A polynomial-time algorithm based on this criterion is given by Golumbic et al. [GMR06] to decide whether a given CNF or DNF is read-once. The algorithm also computes the unique read-once representation when a read-once function is given as input. For $k \geq 2$, no characterization is known for a given monotone Boolean CNF or DNF to be read- k , and in fact, Golumbic et al. asked in [GMR06] whether there exists a polynomial-time algorithm, which given a (normal) monotone Boolean function f in CNF or DNF form, checks whether f is a read- k function, for a fixed k .

The case when the function is given by an oracle has also been considered in the machine learning community. It is shown in [AHK93] that given a read-once function by a membership oracle, we can compute its read-once representation in polynomial time. However, the correctness of the algorithm is based on the assumption that the function provided as an oracle is read-once. If its not read-once then the algorithm terminates with incorrect output.

In this paper, we show that, given an $\wedge - \vee$ -formula, it is NP-hard to check if it represents a read-twice function f . This partially answers the question of Golumbic et al. [GMR06], but leaves open the case when f is given by the CNF or DNF normal form. It follows also from our reduction that it is NP-hard to approximate the readability of a given monotone Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ within a factor of $\mathcal{O}(n)$.

It follows from a result in [Weg87] that almost all monotone Boolean functions on n variables, in which each minterm has size exactly k , have readability $\Omega(n^{k-1} \log^{-1} n)$. Assuming that the function is given by its irredundant DNF (or CNF) of m minterms, this implies a lower bound of $\tilde{\Omega}(m^{1-\frac{1}{k}})$ on the readability. This naturally raises the question whether this bound is tight, i.e. for any monotone CNF formula of m terms, there exists an equivalent read- $\mathcal{O}(m^{1-\frac{1}{k}})$ representation. In this paper, we show that this indeed the case, and moreover that such a representation can be found in polynomial time. In fact, we prove a more general result. For integers $p, q > 0$, let us say a monotone CNF f has (p, q) -bounded intersection [KBEG07] if every p terms intersect in at most q variables. We show that any such CNF has read- $\mathcal{O}((p + q - 1)m^{1-\frac{1}{p+q-1}})$ representation which can be found in polynomial-time. Confronted with this almost tight sublinear bound on readability, an interesting question is whether it can be improved for interesting special cases. For the class of interval DNF's, i.e. those for which there is an ordering on the variables such that each term contains only consecutive variables in that ordering, we show that readability is at most $\mathcal{O}(\log^2 m)$.

The paper is organized as follows. In the next section, we point out that the characterization of [Gur77] for read-once functions does not carry over to read- k functions already for $k = 2$. In Section 3, we present upper bounds on the

readability of some classes monotone Boolean DNF (resp. CNF) that depends only on the number of terms in the normal form. In Section 4 we show that finding the readability in general is hard when the input formula is not a DNF or CNF. We also give an $\mathcal{O}(n)$ inapproximability result in this case.

2 On Generalization of Read-Once Functions

An elegant characterization of read-once functions is provided by the following theorem of Gurvich.

Theorem 1 ([Gur77]). *For any monotone Boolean function f the following two statements are equivalent: (i) f is read-once. (ii) Every minterm and maxterm of f intersect in exactly $c = 1$ variable.*

However, this result does not generalize to read-twice functions as the following example shows. Consider the read-twice formula

$$g(x_1, \dots, x_n, y_1, \dots, y_n) = \bigwedge_{1 \leq i \leq n} (x_i \vee y_i) \bigwedge (x_1 \vee \dots \vee x_n).$$

It is easy to see that the g has a minterm $x_1 \dots x_n$ which intersects with the maxterm $(x_1 \vee \dots \vee x_n)$ in n variables. Hence hypergraphs corresponding to read-twice functions do not necessarily satisfy the generalization of Condition (ii) of Theorem 1 for any constant $c > 1$. Conversely, any such generalization is also not sufficient for a function to be read- c , as implied by the following result on the shortest possible size of k -homogeneous DNF where the size of each term is exactly k (and hence each minterm and maxterm intersect in at most k).

Theorem 2 (cf. [Weg87]). *For an integer k , let $\mathcal{H}_k^n$ be the class of monotone Boolean functions on n variables such that size of every minterm is exactly k . The monotone formula size of almost all $h \in \mathcal{H}_k$ is $\Omega(n^k \log^{-1} n)$.*

Theorem 2 implies that the readability of almost all $h \in \mathcal{H}_k$ is $\Omega(n^{k-1} \log^{-1} n)$, since otherwise the formula achieving a smaller readability has smaller then shortest possible size.

3 Upper Bounds

In this section, we consider various classes of monotone Boolean DNF's and give upper bounds on their readability. First we consider Interval DNF's whose terms correspond to consecutive variables, given some ordering on variables. Next, we consider (p, q) -intersecting DNF where every p of its terms intersect in at most q variables and give an almost tight upper bound on their readability. Finally, we consider a special case of the latter class, namely k -DNF, where the size of each term is bounded by k and again give a tight upper bound on their readability. Even though we get the same upper bound implied by the more general case, the formula computed by our algorithm has only depth 3 in this case.

In our description of the algorithms, we use set-theoretic notations to describe various operations on the structure of DNF's. In this sense, we treat the DNF $\phi = \bigvee t_i$ as its corresponding hypergraph $\{t_i \mid t_i \text{ is a term in } \phi\}$. For example, we write $t \in \phi$ when t is term of ϕ and similarly by $x \in t$ we mean that the term t contains variable x . Let us denote the *degree* of a variable in ϕ by $\deg_\phi(x)$, which is the number of terms in ϕ containing $x \in V$. For a Boolean formula f and a literal x (resp. set of literals S) in f , we denote by $f|_{x=1}$ (resp. $f|_{S=1}$) the resulting f after replacing every occurrence of x (resp. $x \in S$) in f with 1.

3.1 Interval DNF

A monotone Boolean DNF $\mathcal{I} = \bigvee_{I \in \mathcal{I}} \bigwedge_{x \in I} x$ is called interval DNF if there is an ordering of variables $V = \{x_1, x_2, \dots, x_n\}$ such that each $I \in \mathcal{I}$ contains only consecutive elements from the ordering. We show that an interval DNF containing m terms is $\mathcal{O}(\log^2 m)$ -readable. For a variable $x_j \in V$, let $\mathcal{I}_{<x_j} = \{I \in \mathcal{I} : I \subseteq \{x_1, \dots, x_{j-1}\}\}$, $\mathcal{I}_{>x_j} = \{I \in \mathcal{I} : I \subseteq \{x_{j+1}, \dots, x_n\}\}$ and $\mathcal{I}_{\ni x_j} = \{I \in \mathcal{I} : x_j \in I\}$. For a term $I = x_i x_{i+1} \dots x_j$ in interval DNF $\mathcal{I}$, we call x_i and x_j its left and right end-points, and denote them with $L(I)$ and $R(I)$ respectively. We also denote the first (resp. last) term in the ordering of terms of $\mathcal{I}$ with respect to their left end point as $\text{first}(\mathcal{I})$ and $\text{last}(\mathcal{I})$ respectively.

The algorithm is given in Figure 1. It proceeds by choosing a variable x_j such that at most half of the intervals are completely on the left ($\mathcal{I}_{<x_j}$) and half on the right ($\mathcal{I}_{>x_j}$). The formulae for $\mathcal{I}_{<x_j}$ and $\mathcal{I}_{>x_j}$ are computed recursively and the remaining terms in $\mathcal{I}_{\ni x_j}$ are divided into two halves ($\mathcal{I}_1$ and $\mathcal{I}_2$) by considering them in order with respect to their left end-point. The algorithm then factors out common variables from $\mathcal{I}_1$ and $\mathcal{I}_2$ and computes their equivalent formulae recursively.

Theorem 3. *Let $\mathcal{I}$ be an irredundant interval DNF containing m terms. Then $\mathcal{I}$ is $\mathcal{O}(\log^2 m)$ -readable.*

Proof. We show that the procedure $\text{REDUCE1}(\mathcal{I})$ returns a formula with $\mathcal{O}(\log^2 m)$ -readability given an interval DNF. Let $r_1(m)$ and $r_2(m)$ be the readability of the formulae generated by the procedures $\text{REDUCE1}(\mathcal{I})$ and $\text{REDUCE2}(\mathcal{I})$, respectively, when given an interval DNF $\mathcal{I}$ containing m terms as input. Let x_j be the variable chosen in Step 2 of the algorithm. Since the subproblems $\mathcal{I}_{<x_j}$ and $\mathcal{I}_{>x_j}$ are disjoint and have size at most $m/2$, the recurrence for readability computed by $\text{REDUCE1}(\mathcal{I})$ is $r_1(m) \leq r_1(m/2) + r_2(m)$.

Similarly, given an intersecting interval DNF $\mathcal{I}$ the procedure $\text{REDUCE2}(\mathcal{I})$ divides the problem into subproblems $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. Note that the subproblems in the recursive call i.e. $(\mathcal{I}_1 \setminus \{\text{first}(\mathcal{I}_1), \text{last}(\mathcal{I}_1)\})|_{\phi_1=1}$ and $(\mathcal{I}_2 \setminus \{\text{first}(\mathcal{I}_2), \text{last}(\mathcal{I}_2)\})|_{\phi_2=1}$ are again intersecting since $\mathcal{I}_1$ and $\mathcal{I}_2$ are irredundant. For calculating the readability of the formula computed by $\text{REDUCE2}(\mathcal{I})$, consider the case when a variable x_i occurs in both subproblems. We show that if x_i does not occur in ϕ_1 (resp. ϕ_2) then it is necessarily the case that it appears in ϕ_2 (resp. ϕ_1) and thus occurring only once in at least one of the subproblems. Note that since $\mathcal{I}$ is irredundant, the set ϕ_2 forms the interval

Procedure REDUCE1($\mathcal{I}$):

Input: A monotone Boolean interval DNF $\mathcal{I} = \bigvee_{j=1}^m I_j$ on variables
ordering $x_1, \dots, x_n$

Output: A $\mathcal{O}(\log^2 m)$ readable formula ψ equivalent to $\mathcal{I}$

1. **if** $|\mathcal{I}| \leq 2$ **then return** the read-once formula representing $\mathcal{I}$
2. Let $x_j \in V$ such that $|\mathcal{I}_{< x_j}| \leq \frac{m}{2}$ and $|\mathcal{I}_{> x_j}| \leq \frac{m}{2}$
3. **return** $\text{REDUCE1}(\mathcal{I}_{< x_j}) \vee \text{REDUCE2}(\mathcal{I}_{\ni x_j}) \vee \text{REDUCE1}(\mathcal{I}_{> x_j})$

Procedure REDUCE2($\mathcal{I}$):

Input: A monotone Boolean interval DNF $\mathcal{I}$ s.t. $\bigcap_{j=1}^m I_j \neq \emptyset$

Output: A $\mathcal{O}(\log m)$ readable formula ψ equivalent to $\mathcal{I}$

1. **if** $|\mathcal{I}| \leq 2$ **then return** the read-once formula representing $\mathcal{I}$
2. Consider terms in $\mathcal{I}$ in order of their left end points, let $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) be first half (resp. remaining half) elements of $\mathcal{I}$.
3. Let ϕ_1 (resp. ϕ_2) be maximum set of variables that occur in every term of $\mathcal{I}_1$ (resp. $\mathcal{I}_2$)
4. $t_1 := \text{first}(\mathcal{I}_1)$, $t_2 := \text{last}(\mathcal{I}_1)$, $t_3 := \text{first}(\mathcal{I}_2)$, $t_4 := \text{last}(\mathcal{I}_2)$
5. $\psi_1 = \text{REDUCE2}((\mathcal{I}_1 \setminus \{t_1, t_2\}) \mid_{\phi_1=1})$, $\psi_2 = \text{REDUCE2}((\mathcal{I}_2 \setminus \{t_3, t_4\}) \mid_{\phi_2=1})$
6. **return** $(\phi_1 \wedge (\psi_1 \vee t_1 \mid_{\phi_1=1} \vee t_2 \mid_{\phi_1=1})) \vee (\phi_2 \wedge (\psi_2 \vee t_3 \mid_{\phi_2=1} \vee t_4 \mid_{\phi_2=1}))$

Fig. 1. An algorithm to find an $\mathcal{O}(\log^2 m)$ -readable formula for interval DNF consisting of m intersecting terms

$[\text{L}(\text{last}(\mathcal{I}_2)), \text{R}(\text{first}(\mathcal{I}_2))]$. Also observe that since x_i occurs in both subproblems and not in ϕ_1 , it must lie in the interval $[\text{R}(\text{first}(\mathcal{I}_1)), \text{R}(\text{last}(\mathcal{I}_1))]$. It is easy to see that the later interval is the subset of ϕ_2 since $\text{R}(\text{last}(\mathcal{I}_1))$ appears before $\text{R}(\text{first}(\mathcal{I}_2))$ in the ordering of variables because of the definition of $\mathcal{I}_1$ and $\mathcal{I}_2$. Also because of the assumption that $\mathcal{I}$ is intersecting, $\text{L}(\text{last}(\mathcal{I}_2))$ appears before $\text{R}(\text{first}(\mathcal{I}_1))$ in the ordering. So the maximum readability of the formula generated by $\text{REDUCE2}(\mathcal{I})$ where $\mathcal{I}$ consists of m terms satisfies $r_2(m) \leq 2 + r_2(m/2)$. Solving the recurrences yields the stated bound on the readability of $\mathcal{I}$. $\square$

3.2 (p, q) -Intersecting DNF

A monotone Boolean DNF is called (p, q) -*intersecting* if every p of its distinct terms intersect in at most q variables. A quadratic DNF for instance is $(2, 1)$ -intersecting and k -DNF, i. e., DNF where the size of each term is bounded by k is $(2, k-1)$ -intersecting. In this section, we give a $(p+q-1)m^{1-\frac{1}{q+1}}$ bound on the readability of (p, q) -intersecting DNF containing m -terms. Theorem 2 implies that this bound is almost tight because by considering $q+1$ -homogeneous DNF containing $m = \Theta(n^{q+1})$ terms we get,

Corollary 1. *For a constant q , let $\mathcal{G}_q$ be the class of monotone Boolean DNF on n variables with m terms such that size of every minterm is exactly $q+1$. The readability of almost all $g \in \mathcal{G}_q$ is $\Omega(m^{1-\frac{1}{q+1}} \log^{-1} n)$.*

Procedure REDUCE3(ϕ, p, q):Input: A monotone Boolean (p, q) -intersecting DNF ϕ on variables set V Output: A $(p + q - 1)m^{1-\frac{1}{q+1}}$ readable formula ψ equivalent to ϕ

1. $\psi := 0, m := |\phi|$
2. **while** $\exists x \in V$ s.t. $\deg_\phi(x) \geq m^{1-\frac{1}{q+1}}$
3. let $\phi_x = \bigvee_{t \in \phi, x \in t} t$
4. $\phi := \phi \setminus \phi_x$
5. **if** $q > 1$ **then**
6. $\psi := \psi \vee (x \wedge \text{REDUCE3}(\phi_x \mid_{x=1}, p, q - 1))$
7. **else**
8. $\psi := \psi \vee (x \wedge (\phi_x \mid_{x=1}))$
9. **return** $\phi \vee \psi$

Fig. 2. An algorithm to find $(p + q - 1)m^{1-\frac{1}{q+1}}$ readable formula for (p, q) -intersecting DNF consists of m terms

Let ϕ be a (p, q) -intersecting monotone Boolean DNF on variables $V = \{x_1, \dots, x_n\}$. The algorithm is given in Figure 2. It works by picking a variable x with high degree in ϕ and recursively computing a formula equivalent to the part of ϕ where x occurs. The algorithm stops when every variable has low degree in the remaining expression. More precisely, for a variable $x \in V$, let ϕ_x be the DNF consisting of terms of ϕ which contain x , i.e. $\phi_x = \bigvee_{t \in \phi, x \in t} t$. Note that if ϕ is (p, q) -intersecting then $\phi_x \mid_{x=1}$ is $(p, q - 1)$ -intersecting DNF, so the algorithm recurs when $q > 1$ and otherwise it returns the read- $(p - 1)$ formula $x \wedge (\phi_x \mid_{x=1})$. The next Theorem bounds the readability of the formula generated by the algorithm.

Theorem 4. *Given a monotone Boolean DNF μ which is (p, q) -intersecting for $p \geq 2, q \geq 1$. The formula $\mu' = \text{REDUCE3}(\mu, p, q)$ is $(p + q - 1)m^{1-\frac{1}{q+1}}$ readable and it is equivalent to μ .*

Proof. The proof is by induction on q . When $q = 1$, the while loop in Step 2 ensure that every variable in ϕ has degree less then $\sqrt{m}$ after the loop ends. Moreover, a read- $(p - 1)$ formula is added to ψ in each iteration of while loop. Since there are at most $\sqrt{m}$ iterations, the formula $\phi \vee \psi$ in Step 9 has readability at most $\sqrt{m} + (p - 1)\sqrt{m}$.

Now assume that the claim is true for $(p, q - 1)$ intersecting DNF, where $q \geq 2$. We prove it for (p, q) -intersecting DNF using similar arguments as in the previous paragraph. After the while loop ends, every variable in the remaining ϕ has degree less then $m^{1-\frac{1}{q+1}}$. Let $m_1, \dots, m_d$ be number of terms removed from ϕ in each iteration of while loop, where d is the number of iterations. Note that d can be bounded from above by $m^{\frac{1}{q+1}}$ since each m_i is at least $m^{1-\frac{1}{q+1}}$. Now, denoting the readability of (p, q) -intersecting DNF on m terms by $r_{p,q}(m)$, we have

$$r_{p,q}(m) \leq m^{1-\frac{1}{q+1}} + \sum_{i=1}^d r_{p,q-1}(m_i) \leq m^{1-\frac{1}{q+1}} + \sum_{i=1}^d \left((p+q-2)m_i^{1-\frac{1}{q}} \right) \quad (1)$$

$$\leq m^{1-\frac{1}{q+1}} + (p+q-2)d \left(\frac{\sum_{i=1}^d m_i}{d} \right)^{1-\frac{1}{q}} \leq (p+q-1)m^{1-\frac{1}{q+1}}, \quad (2)$$

where we apply induction hypothesis to get Equation (1) and use Jensen's inequality to get Equation (2).

The correctness of the procedure is straightforward since the invariant that $\phi \vee \psi$ is equal to μ holds after completion of every iteration. $\square$

Note that the algorithm produces a depth q formula. In the next section we will see that we can do much better in this regard for the a subclass of (p, q) -intersecting DNF, namely the class of DNF where the size of each term is bounded by a constant k .

3.3 k -DNF

A monotone Boolean DNF is called k -DNF if every term in it has size at most k . In this section, we give an algorithm to compute $2km^{1-1/k}$ readable formula of depth three and equivalent to given k -DNF. We need the following definitions.

A *sunflower* with p petals and a *core* Y is a collection of sets $S_1, \dots, S_p$ such that $S_i \cap S_j = Y$ for all $i \neq j$ and none of the sets $S_i - Y$ is empty. We allow the core Y to be empty however, so every pairwise disjoint family of sets constitutes a sunflower.

Lemma 1 (Sunflower Lemma [ER60]). *Let $\mathcal{H} \subseteq 2^V$ be a hypergraph with $m = |\mathcal{H}|$ and size of each edge is bounded by k . If $m > k!p^k$ then $\mathcal{H}$ contains a sunflower with $p+1$ petals.*

Since a sunflower has a straightforward read-once representation, the above lemma immediately gives an upper bound on the readability of k -DNF with m terms. The algorithm works by finding a sunflower with certain minimal size, representing them as read-once formula and recurse on the remaining edges.

Theorem 5. *Let f be a monotone Boolean DNF with m terms such that the size of each term in f is bounded by k then f is $2km^{1-1/k}$ -readable. Moreover, a formula of such readability and depth 3 can be found in polynomial time.*

Proof. Any k -DNF with m terms contains a sunflower of size at least $(m/k!)^{1/k}$ which we remove and recurse on the remaining terms. Let $r(m)$ denote the readability of boolean k -DNF with m terms then the readability of f can be bounded by the recurrence $r(m) \leq 1 + r(m - (m/k!)^{1/k})$ with $r(2) = r(1) = 1$. By using the inequality $k! \leq k^k$ and substituting $r(m) = 2km^{1-1/k}$ in the above recurrence we get $g(k, m) = 2km^{1-\frac{1}{k}} \left(1 - \left(1 - \frac{m^{\frac{1}{k}-1}}{k} \right)^{1-\frac{1}{k}} \right) \geq 1$. Using elementary calculus, it can be proved that for $k \geq 2$ and $m \geq 1$, the function

$g(k, m)$ is monotonically decreasing in m and monotonically increasing in k . Thus the minimum of g is attained when $k = 2$ and m approaches infinity. The minimum value is 1 and hence $r(m) \leq 2km^{1-1/k}$. Finally, we note that the proof of Lemma 1 is constructive and a sunflower of desired size can be computed in time polynomial in number of variables and terms of a DNF. $\square$

4 Hardness and Inapproximability

In this section, we show that finding the readability of a given monotone Boolean formula is NP-hard. The reduction we use is gap-introducing and so it also gives hardness of approximating readability unless $P = NP$. Our reduction is from the well-known NP-complete problem of deciding satisfiability of a given Boolean 3-CNF $\Phi(x_1 \dots x_n) = \bigwedge_{j=1}^m \Phi_j$. For all $i \in [n]$ and $j \in [m]$, let us define new variables y_{ij}, y'_{ij}, z'_{ij} for a literal x_i in clause Φ_j and variables z_{ij}, z'_{ij}, y'_{ij} for a literal $\neg x_i$ in clause Φ_j . Let $\phi(y_{11} \dots y_{nm}, z_{11} \dots z_{nm})$ be the monotone CNF we get from $\Phi(x_1 \dots x_n)$ by substituting y_{ij} for x_i in Φ_j and z_{ij} for $\neg x_i$ in Φ_j such that $\phi(y, z) \equiv \Phi(x)$, for $y_{ij} = x_i$ and $z_{ij} = \neg x_i$, $i \in [n], j \in [m]$. Furthermore, let $I_i = \{j : x_i \in \Phi_j\} \cup \{j : \neg x_i \in \Phi_j\}$, we define

$$\rho(y', z') = \bigwedge_{i=1}^n \left(\bigwedge_{j \in I_i} y'_{ij} \vee \bigwedge_{j \in I_i} z'_{ij} \right), \quad \psi(y, z, y', z') = \bigvee_{x_i \in \Phi_j} y_{ij} z'_{ij} \vee \bigvee_{\neg x_i \in \Phi_j} y'_{ij} z_{ij}.$$

Now consider the following Boolean function

$$f(y, z, y', z') = \left(\phi(y, z) \bigwedge \rho(y', z') \right) \bigvee \psi(y, z, y', z'). \quad (3)$$

Note that the size of f is $15m$, where m is number of clauses in Φ . The next lemma shows that finding the readability of Boolean formula f defined in Equation (3) is equivalent to solving satisfiability for $\Phi(x)$.

Lemma 2. *The monotone Boolean function f in Equation (3) is read-2 if and only if $\Phi(x)$ is satisfiable. It is read-once otherwise.*

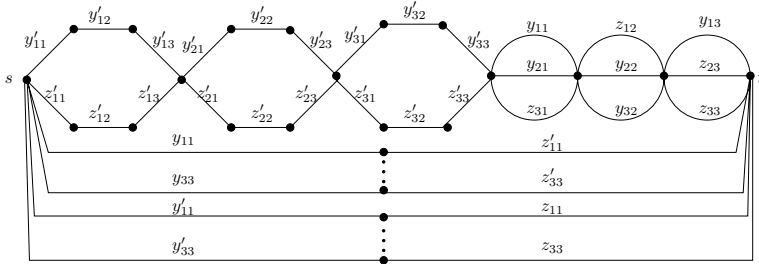


Fig. 3. Applying reduction in Equation (3) to 3-CNF $\Phi = (x_1 \vee x_2 \vee \neg x_3)(\neg x_1 \vee x_2 \vee x_3)(x_1 \vee \neg x_2 \vee \neg x_3)$. Minimal $s - t$ paths in the figure correspond to minterms of f , whereas minimal $s - t$ cuts are maxterms of f .

Proof. Denote the two disjuncts in f by $f_1 = \phi(y, z) \wedge \rho(y', z')$ and $f_2 = \psi(y, z, y', z')$. We first show that the minterms of f_1 which are not absorbed by minterms of f_2 correspond precisely to the satisfiable assignments of Φ and so $f = \psi$ is clearly a read-once function if Φ is not satisfiable.

Let $\hat{x}$ be a satisfiable assignment of $\Phi(x)$. Since $\hat{x}$ makes at least one literal true in each clause of $\Phi(x)$, the set $t_\phi = \{y_{ij} | \hat{x}_i = 1\} \cup \{z_{ij} | \hat{x}_i = 0\}$ contains a minterm t'_ϕ of $\phi(y, z)$. Similarly, note that the set $t_\rho = \{y'_{ij} | \hat{x}_i = 1\} \cup \{z'_{ij} | \hat{x}_i = 0\}$ defines a minterm of $\rho(y', z')$, and so the set $t = t'_\phi \cup t_\rho$ is a minterm of f_1 . It is easy to check that t does not contain any minterm of f_2 since for all $i \in [n]$ and $j \in [m]$, atmost one from each pair y_{ij}, z'_{ij} and y'_{ij}, z_{ij} are members of t .

Conversely, any minterm t of f_1 contains one of $y'_{i1} \dots y'_{im}$ or $z'_{i1} \dots z'_{im}$ for all $i \in [n]$ to cover the conjunct ρ . Assume t is not absorbed by any term of f_2 . Consequently, t does not contain both $y_{ij}z'_{ij}$ or $y'_{ij}z_{ij}$ for all $i \in [n]$ and $j \in [m]$. Therefore it must contain from each clause ϕ_j , at least one of the variable y_{ij} or z_{ij} consistent with the primed variable selected from ρ . Hence the assignment $x_i = 1$ if $y_{ij} \in t$ and $x_i = 0$ if $z_{ij} \in t$ satisfies $\Phi(x)$.

It only remains to prove that f is not a read-once function when $\Phi(x)$ is satisfiable. Assume without loss of generality that the variable x_1 appears in clause Φ_1 . Let us define a maxterm c of f by $c = \{y'_{11}, z'_{11}\} \cup_{i \in [n], j \in [m]} \{y_{ij}, y'_{ij}\}$ and consider the minterm t of f corresponding to a satisfiable assignment $\hat{x}$ of Φ as defined above. It is easy to see that $|t \cap c| > 1$ since for any literal x_i appears in clause Φ_j such that $\hat{x}_i = 1$, t would contain both y_{ij} and y'_{ij} . Hence f is not a read-once function because of Theorem 1. Note that it is read-2 since we have Equation (3) as its read-2 representation. $\square$

Since f in Equation (3) is compose of two read-once formulae, Lemma 2 also implies the hardness of determining if a given monotone formula is disjunction of two read-once formulae.

Corollary 2. *It is NP-hard to know whether a given monotone Boolean formula is a read-once function or a disjunction of two monotone read-once functions.*

Another interesting problem for which we get a hardness result as a corollary of Lemma 2 is the problem of generating all minterms or maxterms of given monotone Boolean formula. Note that the problem can be solved in polynomial time [GG09] when the input formula is read-once.

Lemma 3. *Let $\mathcal{F}$ be the class of monotone Boolean formulae in which each variable appears at most twice. For a formula $f \in \mathcal{F}$, let C and D denote the sets of the maxterms and the minterms of f , respectively.*

(i) *Given a formula $f \in \mathcal{F}$ and a subset C' of C , it is coNP-complete to decide whether $C' = C$.*

(ii) *Similarly, for a formula $f \in \mathcal{F}$ and a subset D' of D , it is coNP-complete to decide whether $D' = D$.*

Proof. Note that since the class $\mathcal{F}$ is closed under duality, both parts of the theorem are equivalent. The hardness of (ii) implied immediately from Lemma 2 by setting $D' = \{t | t \text{ is a term in } \psi\}$. The (possibly) remaining minterms in $D \setminus D'$ correspond to satisfiable assignments of Φ . $\square$

In the following, we generalize the reduction introduced in Equation (3) and get an inapproximability result for the problem of determining readability of given monotone Boolean formula. We use a result of Gál [G02] that gives an explicit monotone Boolean function α on s variables such that the size of the shortest monotone formula representing α is $s^{\Omega(\log s)}$, moreover its irredundant monotone DNF has size $s^{\mathcal{O}(\log s)}$. Note that the readability of α is also $s^{\Omega(\log s)}$, since otherwise we could represent α by a formula with smaller then shortest possible size. We define the following reduction

$$f'(w, y, z, y', z') = \left(\phi(y, z) \bigwedge \rho(y', z') \bigwedge \alpha(w) \right) \bigvee \psi(y, z, y', z'),$$

where the size of f' is $15m + s^{\mathcal{O}(\log s)}$. Note that if Φ is satisfiable, f' has readability $s^{\mathcal{O}(\log s)}$ by applying the same reasoning as in Lemma 2. By choosing s and m such that $m = s^{c_1 \log s}$ and $m = c_2 n$ for a suitable constants c_1, c_2 , we get the following.

Corollary 3. *There is no polynomial-time algorithm to approximate the readability of a given monotone Boolean formula f within factor of $\mathcal{O}(n)$, unless $P = NP$.*

References

- [AHK93] Angluin, D., Hellerstein, L., Karpinski, M.: Learning read-once formulas with queries. J. ACM 40(1), 185–210 (1993)
- [ER60] Erdős, P., Rado, R.: Intersection theorems for systems of sets. J. London Math. Soc. 35, 85–90 (1960)
- [G02] Gál, A.: A characterization of span program size and improved lower bounds for monotone span programs. Comput. Complex. 10(4), 277–296 (2002)
- [GG09] Golumbic, M.C., Gurvich, V.: Read-once functions. In: Crama, Y., Hammer, P.L. (eds.) Boolean Functions: Theory, Algorithms and Applications, Cambridge University Press, Cambridge (in press, 2009)
- [GMR06] Golumbic, M.C., Mintz, A., Rotics, U.: Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. Discrete Applied Mathematics 154(10), 1465–1477 (2006)
- [Gur77] Gurvich, V.: On repetition-free boolean functions. Uspekhi Mat. Nauk (Russian Math. Surveys) 32, 183–184 (1977) (in Russian)
- [KBEG07] Khachiyan, L., Boros, E., Elbassioni, K.M., Gurvich, V.: On the dualization of hypergraphs with bounded edge-intersections and other related classes of hypergraphs. Theor. Comput. Sci. 382(2), 139–150 (2007)
- [Weg87] Wegener, I.: The complexity of Boolean functions. John Wiley & Sons, Inc., New York (1987)

Popular Matchings: Structure and Algorithms

Eric McDermid and Robert W. Irving*

Department of Computing Science, University of Glasgow G12 8QQ, UK
{mcdermid,rwi}@dcs.gla.ac.uk

Abstract. An instance of the *popular matching* problem (POP-M) consists of a set of applicants and a set of posts. Each applicant has a preference list that strictly ranks a subset of the posts. A matching M of applicants to posts is *popular* if there is no other matching M' such that more applicants prefer M' to M than prefer M to M' . This paper provides a characterization of the set of popular matchings for an arbitrary POP-M instance in terms of a structure called the *switching graph*, a directed graph computable in linear time from the preference lists. We show that the switching graph can be exploited to yield efficient algorithms for a range of associated problems, including the counting and enumeration of the set of popular matchings and computing popular matchings that satisfy various additional optimality criteria. Our algorithms for computing such optimal popular matchings improve those described in a recent paper by Kavitha and Nasre [5].

1 Introduction and Background

An instance of the *popular matching problem* (POP-M) consists of a set A of n_1 *applicants* and a set P of n_2 *posts*. Each applicant $a \in A$ has a strictly ordered preference list of the posts in P that she finds *acceptable*. A *matching* M is a set of applicant-post pairs (a, p) such that p is acceptable to a , and each $a \in A$ and $p \in P$ appears in at most one pair in M . If $(a, p) \in M$ we write $p = M(a)$ and $a = M(p)$. An applicant *prefers* a matching M to a matching M' if (i) a is matched in M and unmatched in M' , or (ii) a is matched in both M and M' and prefers $M(a)$ to $M'(a)$. A matching M is *popular* if there is no matching M' such that more applicants prefer M' to M than prefer M to M' . We let $n = n_1 + n_2$, and let m denote the sum of the lengths of the preference lists. It is easy to show that, for a given instance of POP-M, a popular matching need not exist, and if popular matchings do exist they can have different sizes. Abraham et al [1] described an $O(n + m)$ time algorithm for computing a maximum cardinality popular matching, or reporting that none exists.

The results of Abraham et al [1] led to a number of subsequent papers covering variants and extensions of POP-M. See, for example, [2,3,6,7,8,10]. Kavitha and Nasre [5] recently described algorithms to determine an *optimal* popular matching for various interpretations of optimality; in particular they gave an $O(n^2 + m)$

* Both authors supported by EPSRC research grant EP/E011993/1.

time algorithm to find *mincost*, *rank-maximal* and *fair* popular matchings (see Section 3.5 for definitions of these terms).

Our goal in this paper is to characterize the structure of the set of popular matchings for an instance of POP-M, in terms of the so-called *switching graph*. This structure is exploited to yield efficient algorithms for a range of extensions, such as counting and enumerating popular matchings, generating a popular matching uniformly at random, and finding popular matchings that satisfy additional optimality criteria. In particular, we improve on the algorithm of Kavitha and Nasre by showing how mincost popular matchings can be found in $O(n + m)$ time, and rank-maximal and fair popular matchings in $O(n \log n + m)$ time. Detailed proofs of the various lemmas and theorems stated in the subsequent sections of this paper may be found in the full version [9].

The terminology and notation is as in the previous literature on popular matchings (for example, [1,7]). For convenience, a unique *last-resort post*, denoted by $l(a)$, is created for each applicant a , and placed last on a 's preference list. As a consequence, in any popular matching, every applicant is matched, although some may be matched to their last-resort. Let $f(a)$ denote the first-ranked post on a 's preference list; any post that is ranked first by at least one applicant is called an *f-post*. Let $s(a)$ denote the first non-*f*-post on a 's preference list. (Note that $s(a)$ must exist, for $l(a)$ is always a candidate for $s(a)$). Any such post is called an *s-post*. By definition, the sets of *f*-posts and *s*-posts are disjoint.

The following fundamental result, proved in [1], completely characterizes popular matchings, and is key in establishing the structural results that follow.

Theorem 1. (Abraham et al [1]) *A matching M for an instance of POP-M is popular if and only if (i) every f -post is matched in M , and (ii) for each applicant a , $M(a) \in \{f(a), s(a)\}$.*

In light of Theorem 1, given a POP-M instance I we define the *reduced instance* of I to be the instance obtained by removing from each applicant a 's preference list every post except $f(a)$ and $s(a)$. It is immediate that the reduced instance of I can be derived from I in $O(n + m)$ time. For an instance I of POP-M, let M be a popular matching, and let a be an applicant. Denote by $O_M(a)$ the post on a 's reduced preference list to which a is not assigned in M .

2 The Structure of Popular Matchings – The Switching Graph

Given a popular matching M for an instance I of POP-M, the *switching graph* G_M of M is a directed graph with a vertex for each post, and a directed edge (p_i, p_j) for each applicant a , where $p_i = M(a)$ and $p_j = O_M(a)$. A vertex v is called an *f-post vertex* (respectively *s-post vertex*) if the vertex it represents is an *f*-post (respectively *s*-post). We refer to posts and vertices of G_M interchangeably, and likewise to applicants and edges of G_M . An illustrative example of a switching graph for a POP-M instance is given in the full version of this

paper [9]. The example also illustrates each of the forthcoming ideas regarding switching graphs.

A similar graph was defined by Mahdian [6, Lemma 2] to investigate the existence of popular matchings in random instances of POP-M. Note that the switching graph is uniquely determined by a particular popular matching M , but different popular matchings for the same instance yield different switching graphs. The following easily proved lemma gives some simple properties of switching graphs.

Lemma 1. *Let M be a popular matching for an instance I of POP-M, and let G_M be the switching graph of M . Then*

- (i) *Each vertex in G_M has outdegree at most 1.*
- (ii) *The sink vertices of G_M are those vertices corresponding to posts that are unmatched in M , and are all s -post vertices.*
- (iii) *Each component of G_M contains either a single sink vertex or a single cycle.*

A component of a switching graph G_M is called a *cycle component* or a *tree component* according as it contains a cycle or a sink. Each cycle in G_M is called a *switching cycle*. If T is a tree component in G_M with sink p , and if q is another s -post vertex in T , the (unique) path from q to p is called a *switching path*. It is immediate that the cycle components and tree components of G_M can be identified, say using depth-first search, in linear time.

Let C be a switching cycle of G_M . To *apply C to M* is to assign each applicant a in C to $O_M(a)$, leaving all other applicants assigned as in M . We denote by $M \cdot C$ the matching so obtained. Similarly, let P be a switching path of G_M . To *apply P to M* is to assign each applicant a in P to $O_M(a)$, leaving all other applicants assigned as in M . We denote by $M \cdot P$ the matching so obtained.

Theorem 2. *Let M be a popular matching for an instance I of POP-M, and let G_M be the switching graph of M .*

- (i) *If C is a switching cycle in G_M then $M \cdot C$ is a popular matching for I .*
- (ii) *If P is a switching path in G_M then $M \cdot P$ is a popular matching for I .*

Theorem 2 shows that, given a popular matching M for an instance I of POP-M, and the switching graph of M , we can potentially find other popular matchings. Our next step is to establish that this is essentially the only way to find other popular matchings. More precisely, we show that if M' is an arbitrary popular matching for I , then M' can be obtained from M by applying a sequence of switching cycles and switching paths, at most one per component of G_M . First we state a simple technical lemma, the proof of which is an easy consequence of the definition of the switching graph.

Lemma 2. *Let M be a popular matching for an instance I of POP-M, let G_M be the switching graph of M , and let M' be an arbitrary popular matching for I . If the edge representing applicant a in G_M connects the vertex p to the vertex q , then*

- (i) *a is assigned to p in M ;*
- (ii) *if $M'(a) \neq M(a)$ then a is assigned to q in M' .*

Lemmas 3 and 4 consider switching cycles and switching paths respectively.

Lemma 3. *Let M be a popular matching for an instance I of POP-M, let T be a cycle component with cycle C in G_M , and let M' be an arbitrary popular matching for I .*

- (i) *Either every applicant a in C has $M'(a) = M(a)$, or every such a has $M'(a) = O_M(a)$.*
- (ii) *Every applicant a in T that is not in C has $M'(a) = M(a)$.*

Lemma 4. *Let M be a popular matching for an instance I of POP-M, let T be a tree component in G_M , and let M' be an arbitrary popular matching for I . Then either every applicant a in T has $M'(a) = M(a)$, or there is a switching path P in T such that every applicant a in P has $M'(a) = O_M(a)$ and every applicant a in T that is not in P has $M'(a) = M(a)$.*

In terms of the application of switching paths or cycles, separate components of a switching graph behave independently, as captured in the following lemma.

Lemma 5. *Let T and T' be components of a switching graph G_M for a popular matching M , and let Q be either the switching cycle (if T is a cycle component) or a switching path (if T is a tree component) in T . Then, T' is a component in the switching graph $G_{M.Q}$.*

We can now characterize fully the relationship between any two popular matchings for an instance of POP-M.

Theorem 3. *Let M and M' be two popular matchings for an instance I of POP-M. Then M' may be obtained from M by successively applying the switching cycle in each of a subset of the cycle components of G_M together with one switching path in each of a subset of the tree components of G_M .*

An immediate corollary of this theorem is a characterization of the set of popular matchings for a POP-M instance.

Corollary 1. *Let I be a POP-M instance, and let M be an arbitrary popular matching for I with switching graph G_M . Let the tree components of G_M be $X_1, \dots, X_k$, and the cycle components of G_M be $Y_1, \dots, Y_l$. Then, the set of popular matchings for I consists of exactly those matchings obtained by applying at most one switching path in X_i for each i ($1 \leq i \leq k$) and by either applying or not applying the switching cycle in Y_i for each i ($1 \leq i \leq l$).*

3 Algorithms That Exploit the Structure

Each of the algorithms in this section begins in the same way – by constructing the reduced instance, finding an arbitrary popular matching M (if one exists), building G_M , and identifying its components using, say, depth-first search. All of this can be achieved in $O(n + m)$ time. This sequence of steps is referred to as the *pre-processing phase*.

3.1 Counting Popular Matchings

A tree component having q s -posts has exactly $q - 1$ switching paths. For a tree component X_i , denote by $\mathcal{S}(X_i)$ the number of s -posts in X_i . The following theorem is an immediate consequence of Corollary 1.

Theorem 4. *Let I be a POP-M instance, and let M be an arbitrary popular matching for I with switching graph G_M . Let the tree components of G_M be $X_1, \dots, X_k$, and the cycle components of G_M be $Y_1, \dots, Y_l$. Then, the number of popular matchings for I is $2^l * \prod_{i=1}^k \mathcal{S}(X_i)$.*

It follows that an algorithm for counting the number of popular matchings need only carry out the pre-processing phase, counting the number of cycle components and the number of s -posts in each tree component, and all of this can be achieved in linear time.

3.2 Random Popular Matchings

Corollary 1 facilitates the generation of a popular matching for a POP-M instance, uniformly at random, in linear time. The pre-processing phase identifies a popular matching M and the components of G_M . For each cycle component the unique switching cycle is applied or not according to a random bit. For each tree component T , a (possibly empty) switching path is applied according to a random value r in $0, 1, \dots, q - 1$ where q is the number of s -post vertices in T . The algorithm returns the popular matching obtained by applying this choice of switching cycles and switching paths.

3.3 Enumerating Popular Matchings

An algorithm for enumerating the popular matchings begins with the pre-processing phase, during which a popular matching M and the components of G_M are identified. It is then straightforward to enumerate popular matchings by applying or not applying the switching cycle in each cycle component, and applying in turn each switching path (including the empty path) in each tree component. The pre-processing phase occupies $O(n + m)$ time, and the delay in generating each matching is linear in the size of the switching graph, namely $O(n)$.

3.4 Popular Pairs

A *popular pair* for an instance I of POP-M, is an applicant-post pair (a_i, p_j) such that there exists a popular matching M with $(a_i, p_j) \in M$.

Lemma 6. *Let M be a popular matching for an instance I of POP-M. Then, (a_i, p_j) is a popular pair if and only if (i) (a_i, p_j) is in M , or (ii) a_i is an incoming edge to p_j in G_M , and a_i and p_j are in a switching cycle or switching path in G_M .*

It follows from Lemma 6 that the popular pairs can be found in linear time by executing the pre-processing phase followed by a simple traversal of each component of the switching graph.

3.5 Optimal Popular Matchings

Kavitha and Nasre [5] recently studied the following problem: suppose we wish to compute a matching that is not only popular, but is also optimal with respect to some additional well-defined criterion. They defined a natural optimality criterion and described an augmenting path-based algorithm for computing an optimal popular matching. In this section we will describe faster algorithms that exploit the switching graph of the instance to find an optimal popular matching with respect to certain optimality criteria.

For a POP-M instance with n_1 applicants and n_2 posts, we define the *profile* $\rho(M)$ of M to be the $(n_2 + 1)$ -tuple $(x_1, \dots, x_{n_2+1})$ where, for each i ($1 \leq i \leq n_2 + 1$), x_i is the number of applicants who are matched in M with their i^{th} -choice post. The last-resort post is always considered to be the $(n_2 + 1)^{\text{th}}$ -choice post.

Total orders $\succ_R$ and $\prec_F$ on profiles are defined as follows. Suppose that $\rho = (x_1, \dots, x_k)$ and $\rho' = (y_1, \dots, y_k)$. Then

- $\rho \succ_R \rho'$ if, for some j , $x_i = y_i$ for $1 \leq i < j$ and $x_j > y_j$;
- $\rho \prec_F \rho'$ if, for some j , $x_i = y_i$ for $j < i \leq n_2$ and $x_j < y_j$.

A *rank-maximal* popular matching [4] is one whose profile is maximal with respect to $\succ_R$. A *fair* popular matching is one whose profile is minimal with respect to $\prec_F$. (Note that, since the number of $(n_2 + 1)$ th choices is minimised, a fair popular matching is inevitably a maximum cardinality popular matching.) Finally, a *mincost* popular matching is a maximum cardinality popular matching for which $\sum i x_i$ is minimum.

If a *weight* $w(a_i, p_j)$ is defined for each applicant-post pair with p_j acceptable to a_i , then the *weight* $w(M)$ of a popular matching M is $\sum_{(a_i, p_j) \in M} w(a_i, p_j)$. We call a popular matching *optimal* if it is of maximum or minimum weight depending on the context.

With suitable choices of weights, it may be verified that rank-maximal, fair and mincost popular matchings are all examples of optimal popular matchings:

- mincost: assign weight n_2^2 to each pair involving a last resort post, a weight of k to each other pair involving a k^{th} choice, and find a minimum weight popular matching.
- rank-maximal: assign weight $(n_2)^{n_2-k+1}$ to each pair involving a k^{th} choice, and find a maximum weight popular matching.
- fair : assign weight $(n_2)^{k-1}$ to each pair involving a k^{th} choice, and find a minimum weight popular matching.

Kavitha and Nasre [5] described an $O(n^2 + m)$ -time algorithm for finding mincost, rank-maximal and fair popular matchings. In what follows, we give an $O(n+m)$ -time algorithm for finding a mincost popular matching and $O(n \log n + m)$ -time algorithms for finding rank-maximal and fair popular matchings.

We see from the above that very large weights may be assigned to the applicant-post pairs, so we cannot assume that weights can be compared or

added in $O(1)$ time. We assume that the time for comparison or addition of such values is $O(f(n))$ for some function f .

Given an instance of POP-M and a particular allocation of weights, let M be a popular matching, and M_{opt} an optimal popular matching. By Theorem 3, M_{opt} can be obtained from M by applying a choice of at most one switching cycle or switching path per component of G_M . The key is to decide exactly which switching cycles and paths need be applied. In the following, for simplicity of presentation, we assume that “optimal” means “maximum”. Analogous results hold in the “minimum” case.

If T is a cycle component of G_M , an *orientation* of T is either the set of pairs $\{(a, M(a)) : a \in T\}$, or the set $\{(a, M \cdot C(a)) : a \in T\}$, where C is the switching cycle in T . Likewise, if T is a tree component of G_M , an *orientation* of T is either the set of pairs $\{(a, M(a)) : a \in T\}$, or the set $\{(a, M \cdot P(a)) : a \in T\}$, for some switching path P in T . The weight of an orientation is the sum of the weights of the pairs in it, and an orientation of a component is *optimal* if its weight is at least as great as that of any other orientation.

Lemma 7. *If M is an arbitrary popular matching, T is a component of G_M , and M_{opt} is an optimal popular matching, then the set of pairs $\{(a, M_{opt}(a)) : a \in T\}$ is an optimal orientation of T .*

In light of Lemma 7, an algorithm for computing an optimal popular matching can be constructed as follows. For each cycle component T with switching cycle C , an optimal orientation can be found by comparing $\sum_{a \in C} w(a, M(a))$ with $\sum_{a \in C} w(a, M \cdot C(a))$, which is easily achieved in $O(f(n)|T|)$ time. In the case of a tree component T , a depth-first traversal of T can be carried out, starting from the sink, and traversing edges in reverse direction. For an s -post vertex v , the weight of the orientation of T resulting from the application of P_v can easily be found in $O(f(n))$ time from the weight of the orientation resulting from application of P_u , where u is the nearest s -post ancestor of v in the depth-first spanning tree. So the weight of each orientation can be computed in $O(f(n))$ time, and hence an optimal orientation of each tree component T can be found in $O(f(n)|T|)$ time.

Theorem 5. *There is an algorithm to compute an optimal popular matching in $O(m + nf(n))$ time, where n is the number of posts, m is the sum of the lengths of the original preference lists, and $f(n)$ is the maximum time needed for a single comparison of two given weights.*

We use the uniform cost model, which assumes that an arithmetic or comparison operation on numbers of size $O(n)$ has cost $O(1)$. In the case of a mincost popular matching, all weights are $O(n)$, so that, we can take $f(n) = 1$. However, for rank maximal or fair matchings, we can only assume that the weights are $O(n^n)$, so that $f(n) = O(n)$. Hence we have the following corollary.

Corollary 2. (i) *A mincost popular matching can be found in linear time.*
(ii) *A rank-maximal and a fair popular matching can be found in $O(m + n^2)$ time.*

3.6 Improving the Running Time

To improve the complexity of our algorithms for a rank-maximal and a fair popular matching, we discard the weights and work directly with matching profiles. This improved algorithm is described for rank-maximal popular matchings; the changes that need to be made to compute a fair popular matching are similar.

Let Z be a tree component of the switching graph with sink z , let $u \neq z$ be an s-post vertex in Z , and let $v \neq u$ be a vertex such that there is a path $P(u, v)$ in Z from u to v . Any such path $P(u, v)$ is the initial part of the switching path $P(u, z)$ starting at u .

The concept of *profile change* $C(u, v)$ along a path $P(u, v)$ quantifies the effect on the profile of applying the switching path from u , but only as far as v – we call this a *partial switching path*. More precisely, $C(u, v)$ is the sequence of ordered pairs $\langle (i_1, j_1), \dots, (i_r, j_r) \rangle$, where $j_1 < \dots < j_r$, $i_k \neq 0$ for all k , and, for each k , there is a net change of i_k in the number of applicants assigned to their j_k th choice post when $P(u, v)$ is applied.

We define a total order $\succ$ on profile changes (to reflect rank-maximality) in the following way. If $x = \langle (p_1, q_1), \dots, (p_k, q_k) \rangle$ and $y = \langle (r_1, s_1), \dots, (r_l, s_l) \rangle$ are profile changes ($x \neq y$), and j is the maximum index for which $(p_j, q_j) = (r_j, s_j)$, we write $x \succ y$ if and only if

- (i) $k > l$, $j = l$, and $p_{j+1} > 0$; or
- (ii) $k < l$, $j = k$ and $r_{j+1} < 0$; or
- (iii) $j < \min(k, l)$, $q_{j+1} < s_{j+1}$ and $p_{j+1} > 0$; or
- (iv) $j < \min(k, l)$, $q_{j+1} > s_{j+1}$ and $r_{j+1} < 0$; or
- (v) $j < \min(k, l)$, $q_{j+1} = s_{j+1}$ and $p_{j+1} > r_{j+1}$.

A profile change $\langle (i_1, j_1), \dots, (i_r, j_r) \rangle$ is *improving* (with respect to $\succ_R$) if $i_1 > 0$. So an improving profile change leads to a better profile with respect to $\succ_R$. Moreover, if x and y are profile changes with $x \succ y$, and if applying x and y to the same profile ρ yields profiles ρ_x and ρ_y respectively, then $\rho_x \succ_R \rho_y$.

As a next step, we define the following arithmetic operation, which captures the notion of adding an ordered pair to a profile change. For a profile change $C = \langle (i_1, j_1), \dots, (i_r, j_r) \rangle$ and ordered pair (i, j) ($i \neq 0, j > 0$), define $C + (i, j)$ as follows:

$$\begin{aligned} j = j_k, i_k + i \neq 0 &\Rightarrow C + (i, j) = \langle (i_1, j_1), \dots, (i_k + i, j_k), \dots, (i_r, j_r) \rangle. \\ j = j_k, i_k + i = 0 &\Rightarrow \\ C + (i, j) &= \langle (i_1, j_1), \dots, (i_{k-1}, j_{k-1}), (i_{k+1}, j_{k+1}), \dots, (i_r, j_r) \rangle. \\ j_{k-1} < j < j_k &\Rightarrow \\ C + (i, j) &= \langle (i_1, j_1), \dots, (i_{k-1}, j_{k-1}), (i, j), (i_k, j_k), \dots, (i_r, j_r) \rangle. \end{aligned}$$

The algorithm computes an optimal orientation of a tree-component Z by a post-order traversal, viewing Z as rooted at the sink. During this traversal, *processing* a vertex v means determining the best improving profile change C_v obtainable by applying a partial switching path that ends at v , together with the starting vertex u_v of a path $P(u_v, v)$ corresponding to C_v . If no path ending at v has an improving profile change then C_v is null and u_v is undefined.

```

    Traverse( $v$ ) {
        if  $v$  is a leaf
            return null;
        else
            best = null;  start = null;
            for (each child  $w$  of  $v$  that is not an  $f$ -post leaf)
                ( $C_w, u_w$ ) = Traverse( $w$ );
                 $C = C_w + (1, j_w) + (-1, l_w)$ ;          (1)
                if ( $C \succ$  best)                             (2)
                    best =  $C_w$ ;  start =  $u_w$ ;
            return (best, start);  }

```

Fig. 1. The postorder traversal of a tree component

For a leaf vertex v , C_v is trivially null. For a branch node v , C_v and u_v are computed using the best improving profile change C_w for each child w of v in the tree (excluding any such w that is an f -post leaf, since no switching path can begin in such a subtree of v). Let w be a child of v , and let a be the applicant represented by the edge (w, v) of Z . Let posts v and w be the j_w th and l_w th choices, respectively, of applicant a , so that if a were to be re-assigned from post w to post v the profile would gain a j_w^{th} choice and lose an l_w^{th} choice. It follows at once that C_v is determined by the formula

$$C_v = \max\{(C_w + (1, j_w)) + (-1, l_w)\}$$

where the maximum is with respect to $\succ$, and is taken over all children w of v . A pseudocode version of the algorithm appears in Figure 1.

On termination of the traversal, we have determined C_z , the best improving profile change, if any, of a switching path in Z , together with the starting point of such a path. Application of this switching path yields an optimum orientation of Z , or, in case *null* is returned, we know that Z is already optimally oriented.

From the pseudocode in Figure 1, we see that the complexity of the algorithm is determined by the total number of operations involved in steps (1) and (2).

To deal with (1), we represent a profile change by a balanced binary tree B whose nodes contain the pairs (i, j) , ordered by the second member. The $+$ operation on profile changes involves amendment, insertion, or deletion of a node in B , which can be accomplished in time logarithmic in the size of B . Since the number of pairs in a profile change cannot exceed the number of edges in Z , this is $O(\log t)$, and since step (1) is executed at most t times, the total number of operations carried out by step (1), summed over all iterations, is $O(t \log t)$.

As far as (2) is concerned, we first note that two profile changes, involving c_1 and c_2 pairs, with $c_1 < c_2$, can be compared in $O(c_1)$ time. So the cost of a comparison is linear in the size of each of the balanced trees involved. Once a profile change is the ‘loser’ in such a comparison, the balanced tree representing it is never used again. Hence the cost of all such comparisons is linear in s , the sum of the sizes of all of the balanced trees constructed by the algorithm.

But each tree node originates from one or more edges in Z , and each edge in Z contributes to at most one node in one tree. So s is bounded by the number of edges in Z , and hence the total number of operations in step (2), summed over all iterations, is $O(t)$.

It follows that the postorder traversal of a tree component Z with t edges can be completed in $O(t \log t)$ time, and once the optimal switching path is found it can be applied in $O(t)$ time. Hence, since the total number of edges in all tree components is $O(n)$, this process can be applied to all tree components in $O(n \log n)$ time.

Finally, we observe that the optimal orientation of each cycle component can be computed efficiently. For a cycle component Y with switching cycle C , we need only check if the profile change obtained by applying C is an improving profile change, and, if so, C is applied. Hence, the optimal orientation of a cycle component Y with y edges can be computed in $O(y)$ time. This process can therefore be applied to each cycle component in $O(n)$ time. Since the pre-processing phase of the algorithm requires $O(n + m)$ time, we conclude that a rank-maximal popular matching, and by similar means a fair popular matching, can be found in $O(n \log n + m)$ time.

References

1. Abraham, D.J., Irving, R.W., Kavitha, T., Mehlhorn, K.: Popular matchings. *SIAM Journal on Computing* 37, 1030–1045 (2007)
2. Abraham, D.J., Kavitha, T.: Dynamic matching markets and voting paths. In: Arge, L., Freivalds, R. (eds.) *SWAT 2006*. LNCS, vol. 4059, pp. 65–76. Springer, Heidelberg (2006)
3. Huang, C.-C., Kavitha, T., Michail, D., Nasre, M.: Bounded unpopularity matchings. In: Gudmundsson, J. (ed.) *SWAT 2008*. LNCS, vol. 5124, pp. 127–137. Springer, Heidelberg (2008)
4. Irving, R.W., Kavitha, T., Mehlhorn, K., Michail, D., Paluch, K.: Rank-maximal matchings. *ACM Transactions on Algorithms* 2, 602–610 (2006)
5. Kavitha, T., Nasre, M.: Optimal Popular Matchings. In: *Proceedings of MATCH-UP: Matching Under Preferences - Algorithms and Complexity, satellite workshop of ICALP 2008* (2008)
6. Mahdian, M.: Random popular matchings. In: *7th ACM Conference on Electronic Commerce*, pp. 238–242 (2006)
7. Manlove, D.F., Sng, C.T.S.: Popular Matchings in the capacitated house allocation problem. In: Azar, Y., Erlebach, T. (eds.) *ESA 2006*. LNCS, vol. 4168, pp. 492–503. Springer, Heidelberg (2006)
8. McCutchen, R.: The least-unpopularity-factor and least-unpopularity-margin criteria for matching problems with one-sided preferences. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) *LATIN 2008*. LNCS, vol. 4957, pp. 593–604. Springer, Heidelberg (2008)
9. McDermid, E., Irving, R.: Popular Matchings: Structure and Algorithms, Technical Report TR-2008-292, Department of Computing Science, University of Glasgow (November 2008)
10. Mestre, J.: Weighted popular matchings. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) *ICALP 2006*. LNCS, vol. 4051, pp. 715–726. Springer, Heidelberg (2006)

Graph-Based Data Clustering with Overlaps

Michael R. Fellows^{1,*}, Jiong Guo², Christian Komusiewicz^{2,**},
Rolf Niedermeier², and Johannes Uhlmann^{2,***}

¹ PC Research Unit, Office of DVC (Research),
University of Newcastle, Callaghan, NSW 2308, Australia
`michael.fellows@newcastle.edu.au`

² Institut für Informatik, Friedrich-Schiller-Universität Jena
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
`{jiong.guo,c.komus,rolf.niedermeier,johannes.uhlmann}@uni-jena.de`

Abstract. We introduce overlap cluster graph modification problems where, other than in most previous work, the clusters of the target graph may overlap. More precisely, the studied graph problems ask for a minimum number of edge modifications such that the resulting graph consists of clusters (maximal cliques) that may overlap up to a certain amount specified by the overlap number s . In the case of *s-vertex overlap*, each vertex may be part of at most s maximal cliques; *s-edge overlap* is analogously defined in terms of edges. We provide a complete complexity dichotomy (polynomial-time solvable vs NP-complete) for the underlying edge modification problems, develop forbidden subgraph characterizations of “cluster graphs with overlaps”, and study the parameterized complexity in terms of the number of allowed edge modifications, achieving fixed-parameter tractability results (in case of constant s -values) and parameterized hardness (in case of unbounded s -values).

1 Introduction

Graph-based data clustering is an important tool in exploratory data analysis [21,25]. The applications range from bioinformatics [2,22] to image processing [24]. The formulation as a graph-theoretic problem relies on the notion of a *similarity graph*, where vertices represent data items and an edge between two vertices expresses high similarity between the corresponding data items. Then, the computational task is to group vertices into clusters, where a *cluster* is nothing but a dense subgraph (typically, a clique). Following Ben-Dor et al. [2], Shamir et al. [21] initiated a study of graph-based data clustering in terms of *graph modification* problems. More specifically, here the task is to modify (add or delete) as few edges of an input graph as possible to obtain a *cluster graph*,

* Supported by the Australian Research Council. Work done while staying in Jena as a recipient of a Humboldt Research Award of the Alexander von Humboldt Foundation, Bonn, Germany.

** Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.

*** Supported by the DFG, research project PABI, NI 369/7.

that is, a *vertex-disjoint* union of cliques. Numerous recent publications build on this concept of cluster graphs, e.g., [4,7,9,11,13,20]. To uncover the *overlapping* community structure of complex networks in nature and society [18], however, the concept of cluster graphs so far fails to model that clusters may overlap, and it has been criticized explicitly for this lack of overlaps [7]. In this work we introduce a graph-theoretic relaxation of the concept of cluster graphs by allowing, to a certain degree, overlaps between the clusters (which are cliques). We distinguish between “vertex overlaps” and “edge overlaps” and provide a thorough study of the corresponding cluster graph modification problems.

The two core concepts we introduce are *s-vertex overlap* and *s-edge overlap*, where in the first case we demand that every vertex in the cluster graph is contained in at most s maximal cliques and in the second case we demand that every edge is contained in at most s maximal cliques. Clearly, 1-vertex overlap actually means that there is no overlap between the cliques (clusters). Based on these definitions, we study a number of edge modification problems (addition, deletion, editing) in terms of the two overlap concepts, generalizing and extending previous work that focussed on non-overlapping clusters.

Previous work. Perhaps the best studied cluster graph modification problem is the NP-hard CLUSTER EDITING, where one asks for a minimum number of edges to add or delete in order to transform the input graph into a disjoint union of cliques. CLUSTER EDITING has been intensively studied from a theoretical [1,3,9,11,13,20] as well as a practical side [4,7]. The major part of this work deals with the parameterized complexity of CLUSTER EDITING, having led to efficient search-tree based [3,11] and polynomial-time kernelization [9,13,20] algorithms. One motivation of our work is drawn from these intensive studies, motivated by the practical relevance of CLUSTER EDITING and related problems. As discussed before, however, CLUSTER EDITING forces a sometimes too strict notion of cluster graphs by disallowing any overlap. To the best of our knowledge, relaxed versions of CLUSTER EDITING have been largely unexplored. The only approach studying overlapping cliques in the context of CLUSTER EDITING that we are aware of has been presented by Damaschke [6]. He investigated the TWIN GRAPH EDITING problem, where the goal is to obtain a so-called *twin graph* (with a further parameter t specified as part of the input) with a minimum number k of edge modifications. A t -twin graph is a graph whose “critical clique graph” has at most t edges, where the critical clique graph is the representation of a graph that is obtained by keeping for each set of vertices with identical closed neighborhoods exactly one vertex. Roughly speaking, our model expresses a more local property of the target graph. The main result of Damaschke [6] is fixed-parameter tractability with respect to the combined parameter (t, k) . We note that already for $s = 2$ our s -vertex overlap model includes graphs whose twin graphs can have an unbounded number t of edges. Hence, s is not a function of t . Moreover, we expect that for many real-world graphs the number k of necessary edge modifications is much smaller in our model than in the one of Damaschke.

Our results. We provide a thorough study of the computational complexity of clustering with vertex and edge overlaps, significantly extending previous work on CLUSTER EDITING and closely related problems. In particular, in terms of the overlap number s , we provide a complete complexity dichotomy (polynomial-time solvable versus NP-complete) of the corresponding edge modification problems, most of them turning out to be NP-complete (see Table 1 in Section 3). For instance, somewhat surprisingly, whereas CLUSTER EDITING restricted to only allowing edge additions (also known as CLUSTER ADDITION or 1-VERTEX OVERLAP ADDITION) is trivially solvable in polynomial time, 2-VERTEX-OVERLAP ADDITION turns out to be NP-complete. We also study the parameterized complexity of clustering with overlaps. On the negative side, we show W[1]-hardness results with respect to the parameter “number of edge modifications” in case of unbounded overlap number s . On the positive side, we prove that the problems become fixed-parameter tractable for every constant s . This result is based on forbidden subgraph characterizations of the underlying overlap cluster graphs, which may be of independent graph-theoretic interest. Indeed, it turns out that the “1-edge overlap cluster graphs” are exactly the diamond-free graphs. Finally, we develop polynomial-time data reduction rules for two special cases. More precisely, we show an $O(k^4)$ -vertex problem kernel for 1-EDGE OVERLAP DELETION and an $O(k^3)$ -vertex problem kernel for 2-VERTEX OVERLAP DELETION, where both times k denotes the number of allowed edge modifications. We conclude with a number of open problems.

Preliminaries. Given a graph $G = (V, E)$, we use $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G . Let $n := |V|$ and $m := |E|$. The (open) neighborhood $N(v)$ of a vertex v is the set of vertices that are adjacent to v , and the closed neighborhood $N[v] := N(v) \cup \{v\}$. We use $G[V']$ to denote the subgraph of G induced by $V' \subseteq V$, that is, $G[V'] := (V', \{\{u, v\} \mid u, v \in V', \{u, v\} \in E\})$. Moreover, $G - v := G[V \setminus \{v\}]$ for a vertex $v \in V$ and $G - e := (V, E \setminus \{u, v\})$ for an edge $e = \{u, v\}$. For two sets E and F let $E \Delta F := (E \setminus F) \cup (F \setminus E)$ (symmetric difference). For a set X of vertices let $E_X := \{\{u, v\} \mid u, v \in X, u \neq v\}$ denote the set of all possible edges on X . Furthermore, for a graph $G = (V, E)$ and a set $S \subseteq E_V$ let $G \Delta S := (V, E \Delta S)$ denote the graph that results by modifying G according to S . A set of pairwise adjacent vertices is called *clique*. A clique K is a *critical clique* if all its vertices have the same neighborhood and K is maximal. A *graph property* is defined as a nonempty proper subset of the set of graphs closed under graph isomorphism. A *hereditary* graph property is a property closed under taking induced subgraphs.

For a graph property π , the π EDITING problem is defined as follows.

Input: A graph $G = (V, E)$ and an integer $k \geq 1$.

Question: Does there exist a set $S \subseteq V \times V$ with $|S| \leq k$ such that $G \Delta S$ has property π ?

In this paper, we focus attention on π being either the s -vertex overlap property or the s -edge overlap property (see Definition 1 in Section 2). The set S is called a solution. Moreover, we say that the vertices that are incident to an

edge in S are *affected* by S and that all other vertices are *non-affected*. In the corresponding π DELETION (or π ADDITION) problem, only edge deletion (or addition) is allowed.

Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [8,10,17]. One dimension is the input size n (as in classical complexity theory), and the other one is the *parameter* k (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k . This means that when solving a combinatorial problem that is fpt, the combinatorial explosion can be confined to the parameter. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction*. Here, the goal is for a given problem instance x with parameter k to transform it into a new instance x' with parameter k' such that the size of x' is upper-bounded by some function only depending on k , the instance (x, k) is a yes-instance iff (x', k') is a yes-instance, and $k' \leq k$. The reduced instance, which must be computable in polynomial time, is called a *problem kernel*, and the whole process is called *reduction to a problem kernel* or simply *kernelization*.

Downey and Fellows [8] developed a formal framework to show *fixed-parameter intractability* by means of *parameterized reductions*. A parameterized reduction from a parameterized language L to another parameterized language L' is a function that, given an instance (x, k) , computes in $f(k) \cdot n^{O(1)}$ time an instance (x', k') (with k' only depending on k) such that $(x, k) \in L \Leftrightarrow (x', k') \in L'$. The basic complexity class for fixed-parameter intractability is called $W[1]$ and there is good reason to believe that $W[1]$ -hard problems are not fpt [8,10,17].

Due to the lack of space, most proofs are deferred to the full version of this article.

2 Forbidden Subgraph Characterization

In this section, we first introduce the two graph properties considered in this work. Then, we present induced forbidden subgraph characterizations for graphs with these properties.

Definition 1 (*s*-vertex-overlap property and *s*-edge-overlap property). A graph $G = (V, E)$ has the *s*-vertex-overlap property (or *s*-edge-overlap property) if every vertex (or edge) of G is contained in at most s maximal cliques.

Clearly, a graph having the 1-vertex-overlap property consists of a vertex-disjoint union of cliques. See Fig. 1 for a graph fulfilling the 2-vertex-overlap and the 1-edge-overlap property.

Given a graph and a non-negative integer s , we can decide in polynomial time whether G fulfills the *s*-vertex-overlap property using a clique enumeration algorithm with polynomial delay. For each $v \in V$, we enumerate the maximal cliques in $G[N[v]]$. We abort the enumeration if we have found $s + 1$ maximal cliques. Using for example a polynomial delay enumeration algorithm by Makino and Uno [16] that relies on matrix multiplication and enumerates cliques with

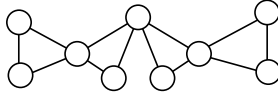


Fig. 1. An example with the 2-vertex overlap and 1-edge-overlap properties

delay $O(n^{2.376})$, the overall running time of this algorithm is $O(s \cdot n^{3.376})$. For the edge case, a similar approach applies. The only difference is that, here, we consider the common neighborhood of the endpoints of every edge, that is, $N[u] \cap N[v]$ for an edge $\{u, v\}$.

Theorem 1. *Given a graph G and a non-negative integer s , it can be decided in time $O(s \cdot n^{3.376})$ (or $O(s \cdot m \cdot n^{2.376})$) time whether G has the s -vertex-overlap (or s -edge-overlap) property.*

The next lemma proves the existence of induced forbidden subgraph characterizations for graphs having the s -vertex-overlap or the s -edge-overlap property.

Lemma 1. *The s -vertex-overlap property and the s -edge-overlap property are hereditary.*

Hereditary graph properties can be characterized by a finite or infinite set of forbidden subgraphs [12]. Thus, by Lemma 1, such a characterization must exist. Here, we show that the forbidden subgraphs have size $O(s^2)$ and, hence, that for fixed s the number of forbidden induced subgraphs is finite. Furthermore, we can find a forbidden induced subgraph in polynomial time.

Theorem 2. *Given a graph G that violates the s -vertex-overlap (or s -edge-overlap) property, one can find in time $O(s \cdot n^{3.376} + s^2 \cdot n)$ (or $O(s \cdot m \cdot n^{2.376} + s^2 \cdot n)$) time a forbidden induced subgraph of size $O(s^2)$.*

See Fig. 2 for the induced forbidden subgraphs for graphs with the 2-vertex-overlap property. Observe that many important graph classes are contained in the class of graphs with the s -overlap property. In particular, it is easy to see that the diamond-free graphs are equivalent to graphs with the 1-edge-overlap property, as stated in Lemma 2. A diamond is the graph that results by deleting one edge from a four-vertex clique. Diamond-free graphs, that is, graphs that contain no diamond as an induced subgraph, form a graph class studied for its own sake [23].

Lemma 2. *A graph G has the 1-edge-overlap property iff G is diamond-free.*

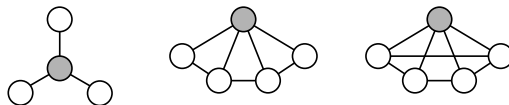


Fig. 2. Forbidden induced subgraphs for the 2-vertex-overlap property. In every graph, the gray vertex is contained in at least three maximal cliques.

Table 1. Classical computational complexity of graph-based data clustering with overlaps. Herein, “NPC” means that the respective problem is NP-complete and “P” means that the problem can be solved in polynomial time.

	s -vertex-overlap	s -edge-overlap
Editing	NPC for $s \geq 1$	NPC for $s \geq 1$
Deletion	NPC for $s \geq 1$	NPC for $s \geq 1$
Addition	P for $s = 1$, NPC for $s \geq 2$	P for $s = 1$, NPC for $s \geq 2$

3 A Complexity Dichotomy with Respect to s

This section provides a complete picture of the computational complexity of the introduced problems. The results are summarized in Table 1.

Lemma 3 shows that if one of the problems is NP-hard for some $s \geq 1$, then it is NP-hard for every $s' \geq s$.

Lemma 3. *For $s \geq 1$, there is a polynomial-time many-one reduction from s -PROPERTY OPERATION to $(s + 1)$ -PROPERTY OPERATION, where PROPERTY $\in \{ \text{VERTEX-OVERLAP}, \text{EDGE-OVERLAP} \}$ and OPERATION $\in \{ \text{EDITING}, \text{DELETION}, \text{ADDITION} \}$.*

Since CLUSTER EDITING and CLUSTER DELETION (equivalent to 1-VERTEX-OVERLAP EDITING and 1-VERTEX-OVERLAP DELETION) are known to be NP-complete [15,21], we directly arrive at the following theorem.

Theorem 3. *s -VERTEX-OVERLAP EDITING and s -VERTEX-OVERLAP DELETION are NP-complete for $s \geq 1$.*

1-VERTEX-OVERLAP ADDITION is trivially polynomial-time solvable: one has to transform every connected component into a clique by adding the missing edges. In contrast, for $s \geq 2$, s -VERTEX-OVERLAP ADDITION becomes NP-complete.

Theorem 4. *s -VERTEX-OVERLAP ADDITION is NP-complete for $s \geq 2$.*

Proof. (Sketch) We present a polynomial-time many-one reduction from the NP-complete MAXIMUM EDGE BICLIQUE problem [19] to 2-VERTEX-OVERLAP ADDITION (2-VOA). Then, for $s \geq 2$, the NP-hardness follows directly from Lemma 3. The decision version of MAXIMUM EDGE BICLIQUE is defined as follows: Given a bipartite graph $H = (U, W, F)$ and an integer $l \geq 0$, does H contain a biclique with at least l edges? A biclique is a bipartite graph with all possible edges.

The reduction from MAXIMUM EDGE BICLIQUE to 2-VOA works as follows: Given a bipartite graph $H = (U, W, F)$, we construct a graph $G = (V, E)$, where $V := U \cup W \cup \{r\}$ and $E := E_{\overline{F}} \cup E_r \cup E_U \cup E_W$. Herein,

- $E_{\overline{F}} := \{\{u, w\} \mid u \in U, w \in W\} \setminus F$,
- $E_X := \{\{x, x'\} \mid x, x' \in X, x' \neq x\}$ for $X \in \{U, W\}$, and
- $E_r := \{\{r, x\} \mid x \in U \cup V\}$.

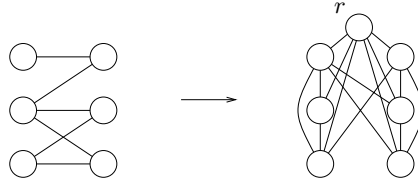


Fig. 3. Example for the reduction from MAXIMUM EDGE BICLIQUE (left graph) to 2-VERTEX-OVERLAP ADDITION (right graph)

That is, the graph $(U, V, E_{\overline{F}})$ is the bipartite complement of H , in G both U and V are cliques, and r is adjacent to all other vertices in G . See Fig. 3 for an illustration of this construction. The correctness proof is deferred to the full version of this article. $\square$

Next, we consider the edge overlap case. First, observe that the reduction given in the proof of Theorem 4 can be easily modified to show the NP-hardness of 2-EDGE-OVERLAP ADDITION: Simply replace the introduced vertex r by an edge e and connect both endpoints of e to all vertices in the given bipartite graph of the MAXIMUM EDGE BICLIQUE. The correspondence between the solutions of both instances can be shown in complete analogy with the vertex overlap case. Note that 1-EDGE-OVERLAP ADDITION is trivially solvable in polynomial time, since there exists only one possibility to destroy a diamond by adding edges; by Lemma 2, diamonds are the only forbidden subgraph of graphs having the 1-edge-overlap property.

Theorem 5. *s -EDGE-OVERLAP ADDITION is NP-complete for $s \geq 2$.*

Finally, we can show that 1-EDGE-OVERLAP EDITING and DELETION are NP-complete by a reduction from VERTEX COVER in cubic graphs. For $s > 1$, the NP-hardness follows directly by using Lemma 3.

Theorem 6. *s -EDGE-OVERLAP EDITING and s -EDGE-OVERLAP DELETION are NP-complete for $s \geq 1$.*

4 Parameterized Complexity

Here, we consider the parameterized complexity of our overlap clustering problems. First, due to Theorem 2, we have a set of forbidden subgraphs for both properties whose size only depends on s . Thus, using a result of Cai [5], we can conclude that all three problems with overlap properties are fixed-parameter tractable with respect to the combined parameter (s, k) .

Theorem 7. *π EDITING, π ADDITION, and π DELETION problems with $\pi \in \{s\text{-VERTEX-OVERLAP}, s\text{-EDGE-OVERLAP}\}$ are fixed-parameter tractable with respect to the combined parameter (s, k) .*

Next, we consider the parameterization with only k as the parameter. This means that s can have an unbounded value. To show $W[1]$ -hardness, we develop a parameterized reduction from the $W[1]$ -complete SET PACKING problem [8].

Theorem 8. *s -VERTEX(EDGE)-OVERLAP DELETION(EDITING) is $W[1]$ -hard with respect to the parameter k in the case of unbounded s .*

5 Two Kernelization Results for Edge Deletion

Not surprisingly, all nontrivial overlap clustering problems we study here seem to become significantly more demanding than clustering without overlaps. Hence, to start with, we subsequently present two kernelization results for the two most basic NP-hard clustering problems with nontrivial overlaps. We defer the correctness proofs of the considered data reduction rules to the full version of this article. It is easy to see that they can be executed in polynomial time.

First, we present a kernelization for 1-EDGE OVERLAP DELETION, which, by Lemma 2, is equivalent to the problem of destroying *diamonds* by at most k edge deletions. We introduce four data reduction rules for this problem and show that an instance that is reduced with respect to all these rules has $O(k^4)$ vertices.

Rule 1. If there is a maximal clique C containing only edges which are not in any other maximal clique, then remove all edges of C .

Rule 2. If there is a matching of size greater than k in the complement graph of the graph that is induced by the common neighbors of the two endpoints of an edge e , then remove e , add e to the solution, and decrease the parameter k by one.

Rule 3. Remove all vertices that are not in any diamond.

Rule 4. If there is a critical clique with more than $k + 2$ vertices, then remove vertices until only $k + 2$ vertices remain.

Theorem 9. 1-EDGE OVERLAP DELETION admits a problem kernel with $O(k^4)$ vertices which can be found in $O(m^3\sqrt{n} + m^2n^2)$ time.

Proof. Let G denote an input graph reduced with respect to the four reduction rules. Partition the vertices of the graph G' , resulting by applying a solution S to the input graph G , into two subsets, one set X containing the vertices that are endpoints of edges deleted by S and $Y := V \setminus X$. Further, construct for each edge $e \in S$ a set Y_e containing the vertices in Y that occur together with e in some diamonds. By Rule 3, $Y = \bigcup_{e \in S} Y_e$.

First, we show that for every maximal clique C in $G'[Y]$ it holds that $C \subseteq Y_e$ for an edge $e \in S$: In G , there is a maximal clique C' containing C and, by Rule 1, C' has an edge $e = \{u, v\}$ which is in two maximal cliques. If $e \in S$, then every vertex in C should be in Y_e ; otherwise, there must be a vertex $w \notin C'$ that is adjacent to both u and v . One of the edges $\{u, w\}$ and $\{v, w\}$ must then be in S . Thus, every vertex in C must build a diamond with this deleted edge and C is contained in either $Y_{\{u, w\}}$ or $Y_{\{v, w\}}$.

Next, we prove that, for every edge $e = \{u, v\} \in S$, at most $4k$ maximal cliques of $G'[Y]$ are subsets of Y_e . Clearly, all vertices in Y_e must be adjacent to one of u and v . Let $N_{u,v}$ denote the common neighbors of u and v in Y_e . Obviously, $N_{u,v}$ is an independent set and, by Rule 2, $|N_{u,v}| \leq 2k$. Let $N_v := (N(v) \setminus N(u)) \cap Y_e$ and $N_u := (N(u) \setminus N(v)) \cap Y_e$. Since $N_{u,v}$ is an independent set, no vertex from $N_v \cup N_u$ can be adjacent to two vertices in $N_{u,v}$. Then, we can partition the vertices in $N_u \cup N_v$ into at most $4k$ subsets according to their adjacency to the vertices from $N_{u,v} = \{x_1, \dots, x_l\}$ with $l \leq 2k$, every subset N_{u,x_i} (or $N(v, x_i)$) containing the vertices in $N(u) \cap N(x_i)$ (or $N(v) \cap N(x_i)$). It is easy to see that each of these subsets is a clique, since, otherwise, we would have some undestroyed diamond. With the same argument, there cannot be an edge between N_{u,x_i} and N_{u,x_j} with $i \neq j$. Moreover, the edges between N_{u,x_i} and N_{v,x_j} , if there are any, do not belong to the maximal cliques that are contained in Y_e . The reason is that the two endpoints of such an edge cannot have common neighbors in Y_e ; otherwise, there would be some undestroyed diamond. Thus, we have at most $4k$ maximal cliques in $G'[Y]$ which are entirely contained in Y_e .

Finally, we show that if two vertices $u, v \in Y$ are contained in exactly the same sets of maximal cliques in Y , then they have the same neighborhood in G . Assume that this is not true. Then, u and v must have different neighborhoods in X . Let $w \in X$ be a neighbor of u but not of v . Since every two maximal cliques in Y can intersect in at most one vertex (due to the 1-edge overlap property), there can be only one maximal clique in Y containing both u and v . Assume that this maximal clique is contained in Y_e for an edge $e \in S$. Moreover, there must be another clique C in G containing w and u , but not v . By Rule 1, C must contain an edge which is part of two maximal cliques. This implies that the vertices w and u have to be in $Y_{e'}$ for an edge $e' \in S$ and $e \neq e'$. This means that there has to be a maximal clique in $Y_{e'}$ containing u but not v , contradicting that u and v are contained in the same sets of maximal cliques in Y .

Putting all the arguments together, we can now show an upper bound for the number of vertices in the reduced instance. Clearly, $|X| \leq 2k$. To bound $|Y|$, note that we have at most k Y_e 's. Each of them contains at most $4k$ maximal cliques of $G'[Y]$. Since every maximal clique of $G'[Y]$ is contained in Y_e for one $e \in S$, we have altogether at most $4k^2$ maximal cliques in $G'[Y]$. It remains to show a size bound for each of these cliques. From the vertices in one clique K , only $4k^2$ of these can be in more than one maximal clique in Y , since every two such cliques overlap in at most one vertex. The remaining vertices of K then have identical neighborhoods. Thus, by Rule 4, K contains at most $4k^2 + k + 2$ vertices. This yields the required size bound on $|Y|$ and, therefore, on the reduced instance. $\square$

Next, we provide a kernelization for 2-VERTEX OVERLAP DELETION. In the following, we say that a vertex is *satisfied* if it is contained in at most two maximal cliques, and a clique is satisfied if all its vertices are satisfied. A clique is a neighbor of an other clique if they share some vertex or edge. Here, the polynomial-time executable data reduction rules read as follows.

Rule 1. If there is a critical clique K with more than $k + 1$ vertices, then remove vertices from K until only $k + 1$ vertices remain.

Rule 2. If there exists a satisfied maximal clique K and K 's neighbors are all satisfied, then remove all edges in K that are not in other maximal cliques.

Rule 3. Let G be a graph reduced with respect to Rule 1. Let K be a maximal clique of G . Consider ℓ maximal cliques $K_1, \dots, K_\ell$ fulfilling the following two conditions:

- 1.) $K \cap K_i \neq \emptyset$, $1 \leq i \leq \ell$, and
- 2.) all vertices in K_i are satisfied, $1 \leq i \leq \ell$.

If $\sum_{i=1}^{\ell} |K_i \cap K| \geq 3k + 4$, then delete all edges between $K_1 \cap K$ and $K \setminus K_1$.

Rule 4. Remove connected components that fulfill the 2-vertex overlap property.

Theorem 10. 2-VERTEX OVERLAP DELETION admits a problem kernel with $O(k^3)$ vertices.

6 Conclusion

We have studied for the first time new cluster graph modification problems motivated by the practical relevance of clustering with overlaps [7,18]. Naturally, studying a so far unexplored set of problems, there remain many challenges for future work. We list only a few of them. First, it is conceivable that the forbidden subgraph characterizations we developed for cluster graphs with overlaps can be further refined. Second, it is desirable to improve the upper bounds on our fixed-parameter algorithms (including the kernelization results) and to further extend the list of fixed-parameter tractability results (in particular, achieving kernelization results for problems other than 1-EDGE OVERLAP DELETION and 2-VERTEX-OVERLAP DELETION). Third, corresponding experimental studies (like those undertaken for CLUSTER EDITING, see [4,7]) are a natural next step. Fourth, the polynomial-time approximability of our problems remains unexplored. Fifth and finally, it seems promising to study overlaps in the context of the more general correlation clustering problems (see [1]) or by relaxing the demand for (maximal) cliques in cluster graphs by the demand for some reasonably dense subgraphs (as previously considered for CLUSTER EDITING [14]).

References

1. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. *Mach. Learn.* 56(1-3), 89–113 (2004)
2. Ben-Dor, A., Shamir, R., Yakhini, Z.: Clustering gene expression patterns. *J. Comput. Biol.* 6(3/4), 281–292 (1999)
3. Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: Going weighted: Parameterized algorithms for cluster editing. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) *COCOA 2008*. LNCS, vol. 5165, pp. 1–12. Springer, Heidelberg (2008)
4. Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: Evaluation and experiments. In: McGeoch, C.C. (ed.) *WEA 2008*. LNCS, vol. 5038, pp. 289–302. Springer, Heidelberg (2008)

5. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.* 58(4), 171–176 (1996)
6. Damaschke, P.: Fixed-parameter enumerability of Cluster Editing and related problems. *Theory Comput. Syst.* (to appear, 2009)
7. Dehne, F., Langston, M.A., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The cluster editing problem: Implementations and experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) *IWPEC 2006*. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006)
8. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)
9. Fellows, M.R., Langston, M.A., Rosamond, F.A., Shaw, P.: Efficient parameterized preprocessing for Cluster Editing. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) *FCT 2007*. LNCS, vol. 4639, pp. 312–321. Springer, Heidelberg (2007)
10. Flum, J., Grohe, M.: *Parameterized Complexity Theory*. Springer, Heidelberg (2006)
11. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. *Theory Comput. Syst.* 38(4), 373–392 (2005)
12. Greenwell, D.L., Hemminger, R.L., Klerlein, J.B.: Forbidden subgraphs. In: *Proc. 4th Southeastern Conf. on Comb., Graph Theory and Computing*, *Utilitas Mathematica*, pp. 389–394 (1973)
13. Guo, J.: A more effective linear kernelization for Cluster Editing. *Theor. Comput. Sci.* 410(8–10), 718–726 (2009)
14. Guo, J., Komusiewicz, C., Niedermeier, R., Uhlmann, J.: A more relaxed model for graph-based data clustering: *s*-plex editing. In: *Proc. 5th AAIM*. LNCS, Springer, Heidelberg (2009)
15. Krivánek, M., Morávek, J.: NP-hard problems in hierarchical-tree clustering. *Acta Inform* 23(3), 311–323 (1986)
16. Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques. In: Hagerup, T., Katajainen, J. (eds.) *SWAT 2004*. LNCS, vol. 3111, pp. 260–272. Springer, Heidelberg (2004)
17. Niedermeier, R.: *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, Oxford (2006)
18. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435(7043), 814–818 (2005)
19. Peeters, R.: The maximum edge biclique problem is NP-complete. *Discrete Appl. Math.* 131(3), 651–654 (2003)
20. Protti, F., da Silva, M.D., Szwarcfiter, J.L.: Applying modular decomposition to parameterized cluster editing problems. *Theory Comput. Syst.* 44(1), 91–104 (2009)
21. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. *Discrete Appl. Math.* 144(1–2), 173–182 (2004)
22. Sharan, R., Maron-Katz, A., Shamir, R.: CLICK and EXPANDER: a system for clustering and visualizing gene expression data. *Bioinformatics* 19(14), 1787–1799 (2003)
23. Talmaci, M., Nechita, E.: Recognition algorithm for diamond-free graphs. *Informatica* 18(3), 457–462 (2007)
24. Wu, Z., Leahy, R.: An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(11), 1101–1113 (1993)
25. Xu, R., Wunsch II, D.: Survey of clustering algorithms. *IEEE Transactions on Neural Networks* 16(3), 645–678 (2005)

Directional Geometric Routing on Mobile Ad Hoc Networks

Kazushige Sato and Takeshi Tokuyama

GSIS, Tohoku University, Sendai, 980-8579, Japan
{kazushige,tokuyama}@dais.is.tohoku.ac.jp

Abstract. We consider the geometric routing problem on mobile ad-hoc wireless networks such that we can send packets by only using local information at each node. We design a novel routing algorithm named *directional routing algorithm* by using random local neighbors such that the information can be efficiently maintained if each node dynamically changes its position. In our scheme, the number of hops and the number of transmissions issued during the routing process are both very small. No centralized control is necessary: The network is implicitly stored and maintained in a distributed fashion at each node using $O(1)$ space, and constructed only using local information in transmission disk of each node. Using this network, we give the first geometric routing with non-trivial theoretical performance guarantee.

1 Introduction

A mobile ad hoc network (MANET) is a vital infrastructure in recent information society. MANET is a system of wireless mobile nodes dynamically self organizing in arbitrary and temporary network topologies. People and vehicles can thus be internetworked in areas without a pre-existing communication infrastructure, or when the use of such infrastructure requires wireless extension. From the viewpoint of algorithm theory, design of efficient algorithms on networking problems in a MANET is an important research subject. A mathematical model of a wireless network is as follows: Consider a set $S = \{p_1, p_2, \dots, p_n\}$ of *router nodes* of the network represented as a set of n point in the Euclidean plane. Each router node p_i has its transmission radius r_i , and it can send messages to nodes in its transmission area (called *transmission disk* in this paper), which is the disk of radius r_i around p_i . Thus, two router nodes can directly communicate if they are in the transmission disk of each other. We consider a graph that has the vertex set S , and an edge is given between each pair of the two nodes that can directly communicate to each other.

For simplicity, we assume that each router node has the largest transmission radius 1, although we allow each node to adjust the electric power to shrink the transmission radius to save energy and reduce traffic. If the transmission radius is 1 for every node, we call the network the *uniform radius network*, and the corresponding graph is called the *unit-disk graph* and denoted by $UDG(S)$, which is indeed the intersection graph of disks of radius $1/2$ around points of S .

We assume that $\text{UDG}(S)$ is connected, since otherwise no routing strategy can send messages between its different connected components.

Suppose that a node s want to send a packet to a destination node t along a path in (a subgraph H of) $\text{UDG}(S)$. See Figure 1 for example. If s knows a path in $\text{UDG}(S)$ towards t and can attach the information to the packet, it is easy to deliver the packet. This is called a *proactive* routing. However, we consider the *reactive* case where each node only know local information of $\text{UDG}(S)$ within its transmission area and the route is explored when the packet is actually being sent. If the packet is sent without help of the geometric information of nodes, we need to broadcast control messages to search a path in the graph $\text{UDG}(S)$. A typical method is the *flooding* operation, and several routing methods based on flooding have been proposed [10,5,3]; however, flooding is an expensive operation in a MANET.

Fortunately, we can often utilize the geometric location of each node that can be obtained by using a positioning tool provided for the node: Sensor-based positioning system and GPS (Global Positioning System) are examples of such tools. A routing method with the help of geometric location information is called the *geometric routing*. The geometric routing problem is formulated as follows: We send a packet from a given node s to its destination t along a unknown path P in the graph $\text{UDG}(S)$ from s to t . At each node v on the path, the packet is forwarded to its adjacent node according to a protocol. The geometric locations (i.e. coordinate values) of s and t are known, but the information of P is not given in advance. Each node v knows its exact geometric location, and can find locations of nodes in its transmission disk by sending a control message and receive answers. However, the network service quality deteriorates if we issue too many control messages, and we would like to minimize the number of such messages.

In a MANET, each node of the set S can move, and also S can be updated due to insertion, deletion, or node-failure. We call the problem *dynamic* if we handle such update/movement, and maintain the efficient routing mechanism; otherwise *static*. Each node v may store some information of its neighbor nodes, although it might be obsolete in the dynamic setting unless v sends a new control message and confirm.

We note that the position of the destination t must be obtained by some mechanism. If s finds the position of t , we can recorded it in the packet and use it under the assumption that the movement of t during sending a packet is small enough to ignore. A typical case is that t is among a set of small number of nodes whose position is known to everyone. Especially, in a sensor network, the nodes are classified into mobile sensor nodes and a small number of (usually static) information centers, and each sensor node sends packets towards information centers. Exchange of packets between sensor nodes via an information center is also possible if the information center can find location of t . We may also consider a system in which s can query the position of t to a (possibly distributed) database storing recent locations of nodes.

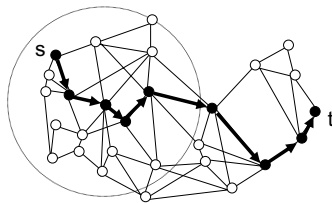


Fig. 1. Routing on a subgraph of $UDG(S)$

A popular approach for the geometric routing is the *greedy routing*, in which the packet is sent to the neighbor of v that is nearest to t . Another interesting method is the compass routing [7] which sends the packet to the neighbor w of v minimizing the angle $\angle twv$. In the original greedy routing, a node v receiving a packet searches for the next node among all its neighbor nodes. That is, v sends a control message to all the nodes in its transmission disk, and every node receiving the message sends back an acknowledgement message containing its location. We call this operation the *neighbor-find operation* in this paper. However, the neighbor-find operation causes one-to-many bidirectional communication, and many acknowledgement messages can be created if there are many nodes in the transmission disk. This is not only expensive in computation and energy consumption but also causes heavy interference of the messages. On the other hand, if v knows the next node w beforehand, we can directly send the packet to w and only w needs to reply acknowledgement to v ; this reduces the traffic a lot.

One possible method to reduce the number of neighbor-find operations is that v maintains the exact location of all its neighbors in its local memory to find the nearest one to the destination t without issuing the control message. However, this needs additional space and computation, and especially consuming in the dynamic situation, since we need to frequently inspect the location of neighbors to update the information. Thus, if there are many nodes in a transmission disk (i.e., the routers are densely distributed), it is better to consider a small number of neighbors, and send the packet to one of them. This defines a connected spanning subgraph H of $UDG(S)$ such that the routing is done on H (see Figure 1). This can also reduce the energy consumption, since we can reduce the transmission radius of v to the distance towards its neighbors of v in H . Especially in the static situation, several graphs such as the Delauney triangulation and the Gabriel graph are considered in the literature.

The greedy routing is easy and usually efficient in number of hops on the route. Unfortunately, the greedy routing (on any H) fails if there is no neighbor node that is nearer to t than v . It is also known that the compass routing may fail to send a packet to t . Thus, we need additional technology to fix this defect. There are several proposals to design a geometric routing that surely send a packet. Bose et al [2] gave a method named *face-routing*, which always correctly send a packet tracing boundaries of faces if H is a planar graph. Kuhn et al [9] proposed the adaptive face-routing method, and showed that their algorithm is optimal

on a worst-case planar graph H . However, a route obtained by a face-routing often needs a lot of hops to reach the destination, and inefficient compared to a route selected by a greedy method.

Thus, a practically better method is a hybrid method of the greedy routing and the (adaptive) face routing: We apply a greedy method, and if it gets stuck, we call an exceptional routine (e.g., face routing) to complete the routing. GOAFR⁺ proposed by Khun et al. [8] is such an algorithm, in which the Gabriel graph is considered as H . One defect about the Gabriel graph is that it has too few edges and it often calls the exceptional routine. Another defect is that it is not easy to maintain the Gabriel graph in the dynamic situation. Moreover, since its edges are generally short, the number of hops in the route tends to be large, and therefore, it is inferior in the number of hops. More seriously, no nontrivial theoretical analysis of the performance of the above existing methods has been given.

In this paper, we propose a new geometric routing scheme that is perfectly reactive, locally constructive with no centralized set-up operation, and also has theoretical performance guarantee. Such a scheme is the first one in ad-hoc network routing research, as far as the authors know. Moreover, it is not only theoretically but also practically efficient. Our scheme uses a *generalized local neighbor graph* obtained by connecting each nodes to its *local neighbors* each of which is selected (by any protocol) from a sextant of the transmission disk $D(v)$ of each node v . The scheme consists of a greedy routing called *directional routing* on the generalized local neighbor graph together with an exceptional routine called *one-face* routing on the *relative neighborhood graph*. It has a feature that it is easier to maintain dynamically and also we can analyze its theoretical performance if we implement the scheme such that each local neighbor is selected randomly from the associated sextant of the transmission disk.

Our directional routing algorithm can be considered as an intermediate of the compass routing and (the original) greedy routing. Indeed, it sends the packet to a neighbor w of v such that the angle $\angle tvw$ is small (not necessarily smallest) and the distance towards the destination is decreased. Since our scheme have a freedom of choice of the local neighbors, we can efficiently maintain the routing connection against dynamic change of locations of nodes. The generalized local neighbor graph is not always strongly connected as a directed graph even for a good point distribution. Although it may look strange to do routing on such a graph, we overcome the lack of strongly connectivity by using the property that a node can find whether t is in its transmission radius by locally computing the distance between t and it (without any network communication).

For a general point distribution, the directional routing may get stuck. Each node on the route can detect whether it needs an exceptional routine to find a detour and continue the routing. The exceptional routine is the one-face routing (that is a variant of face routing that only searches in one face of a planar graph) using the relative neighborhood graph. We theoretically guarantee that our routing scheme never fails. Moreover, we give a novel implementation of the

one-face routing to give an $O(d(s, t))$ upper bound for the number of calls of one-face routing routine, where $d(s, t)$ is the distance between s and t .

Since we have freedom of choice of local neighbors, there are implementation issue that how we should choose the local neighbors in order to obtain an efficient system. For a version of our algorithm in which each local neighbor is selected randomly from the sextant, we give an $O(d(s, t) \log^2 \alpha^{-1})$ bound of the number of hops on the route for the directional routing (ignoring hops in the one-face routing). Here, $\alpha < 1/2$ is a parameter such that the distance between every pair of router nodes is larger than α . We use this parameter to give mathematical analysis of performance of routing schemes, and the analysis holds even if there are small number of pairs violating the distance condition.

As far as the authors know, this is the first nontrivial asymptotic analysis of the number of hops of geometric routings. We note that this is the worst case analysis, and in a uniform distribution, the number of hops is $O(d(s, t))$. We give experimental results to confirm the theoretical analysis and show advantage of our method in both static and dynamic settings. In our scheme, we need to keep $O(1)$ information at each node, and the number of hops is very small, and each hop in the directional routing process is attained in the minimum number of information exchange (i.e., sending the packet from the sender and the acknowledgement from the receiver). Each node only uses information in its transmission disk, and no centralized control is necessary provided that the unit disk graph is connected.

2 Directional Routing Using Generalized Local Neighbor Graph

Let $D(v, r) = \{x \in \mathbb{R}^2 : d(x, v) < r\}$ be the disk of radius r around a point v in the plane, and we write $D(v)$ for the unit disk $D(v, 1)$. The region $P(v, k) = \{p \neq v : (k-1)\pi/3 \leq \arg(vp) < k\pi/3\}$ ($k = 1, 2, \dots, 6$) is called the k -th wedge around v , where $\arg(vp)$ is the argument angle of the vector vp . $Q(v, k) = P(v, k) \cap D(v)$ is called the k -th sextant around v .

We say that the point configuration S is *local-neighbor perfect* if $Q(v, k) \cap S \neq \emptyset$ whenever $P(v, k) \cap S \neq \emptyset$ for any v and k . By definition, any point configuration becomes local-neighbor perfect if the transmission radius is large enough such that the nearest point (if any) of $P(v, k) \cap S$ is in the transmission disk for each v and k .

Given a set S of n points in the plane, we define $F = (f_1, f_2, \dots, f_6)$ such that f_k is a map from S to $S \cup \{*\}$ such that $f_k(v) \in Q(v, k)$ if $Q(v, k) \neq \emptyset$ and $f_k(v) = *$ otherwise. The asterisk symbol means “none”. The 6 points (or possibly $*$) $f_k(v)$ ($k = 1, 2, \dots, 6$) are called the *local neighbors* of v associated with F . The *generalized local neighbor graph* associated with F is a directed graph on the vertex set S with directed edges $(v, f_k(v))$ for each $v \in S$ and $k = 1, 2, \dots, 6$, in which we ignore edges towards $*$. The graph depends on F and is denoted by $\text{GLNG}_F(S)$ formally; however, it is denoted by $\text{GLNG}(S)$ if F is not explicitly considered. It is clear that $\text{GLNG}(S)$ can be constructed in a distributed fashion by only using local information within $D(v)$ for each $v \in S$.

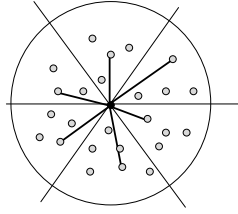


Fig. 2. RLNG(S) neighbors

If S is local neighbor perfect and $f_k(v)$ is the nearest point in $Q(v, k)$ to v , GLNG(S) becomes the *local neighbor graph* LNG(S), which is a popular graphs in computational geometry [12,11,6], and it has been used to design an ad-hoc network with low interference [4]. If we choose $f_k(v)$ randomly from $Q(v, k)$ for each $v \in S$ and $1 \leq k \leq 6$, GLNG(S) is called a random local neighbor graph and denoted by RLNG(S) (See Figure 2).

GLNG(S) is not always strongly connected as a directed graph even if S is local neighbor perfect (note: LNG(S) is strongly connected). One may feel absurd to consider a routing method using GLNG(S); Nevertheless, utilizing the nature of a wireless network and geometric information, we can design the following routing strategy.

Directional routing algorithm

1. Initialize $v = s$
2. If $t \in D(v)$, send the packet to t and complete the process.
3. Find k such that $t \in P(v, k)$. If $f_k(v) = *$, halt;
otherwise send the packet to $f_k(v)$, set $v = f_k(v)$ and go to Step 2.

Note that the last hop (corresponding to Step 2 of the algorithm) in the directional routing need not be an edge of GLNG(S); this restores the connectivity of the network. Suppose that the algorithm does not halt at Step 3, and consider any node v on the route. If $t \in D(v)$, the routing succeeds. Otherwise, the packet is sent to $w = f_k(v)$. Now, consider the distance $d(v, t)$ and $d(w, t)$. Since both are in $P(v, k)$ and $d(v, t) > d(v, w)$, we have $d(v, t) > d(w, t)$. Since the distance strictly decreases and S is a finite set, the packet is sent to t eventually.

If S is local-neighbor perfect, $t \in P(v, k)$ implies $Q(v, k) \neq \emptyset$; therefore, the directional routing algorithm never halts at Step 3, and we have the following:

Theorem 1. *If S is local-neighbor perfect, the directional routing method on GLNG(S) always successfully sends the packet.*

3 Detour Finding If the Directional Routing Halts

Let us consider the case where S is not local-neighbor perfect. The directional routing algorithm may halt by finding $f_k(v) = *$. For the case, we need an

exceptional routine to continue the routing. We note that such a detour-finding routine is inevitable for any known geometric routing strategy.

The relative neighborhood graph is an undirected graph on the vertex set S such that it has an edge (u, v) if and only if there is no node of S in the *lens* $D(u, r) \cap D(v, r)$ where $r = d(u, v)$. We consider the graph $H(S)$ that consists of all the edges shorter than 1 in the relative neighborhood graph of S . The relative neighborhood graph contains a minimum spanning tree (MST) of S [1,11], and accordingly, $H(S)$ contains the MST, since MST only uses edges shorter than 1 if UDG is connected. Also, all the neighbors of a vertex v in $H(S)$ are relative neighbors in $LNG(S)$, and hence the degree is at most 6. We name the following method the *one-face routing* on $H(S)$. An advantage of the method to previous face routing algorithms [2,8] is that we can give a theoretical bound of number of faces traced during sending a packet.

Suppose that the directional routing algorithm halts at v . Let R be the unique face of the planar graph $H(S)$ such that R has v on its boundary and R intersect the segment vt in the neighbor of v . Let ∂R be the boundary chain of R . We trace on ∂R to find a vertex q called *exit point* sufficiently nearer to t than v . We may use the bidirectional tracing method given in [8].

Although the intersection point of ∂R and vt is nearer to t than v , it is not trivial to show that there is a vertex of ∂R nearer to t . Indeed, it is false for a general planar graph, and we need properties of $H(S)$ and our assumption that $UDG(S)$ is (hence, $H(S)$ is) connected. Consider a quadrant Δ of the unit disk $D(v)$ such that vt is the halving halfline of Δ . Let D be the disk of a radius $\max(r - \delta, 1)$ around t , where $r = d(v, t)$ and $\delta \leq 1/2$ is a parameter. Naturally, if we find a point in D , then either the distance towards the target is decreased by δ or t is found in the next step. We can prove the following lemma:

Lemma 1. *If $\Delta \setminus D$ contains no point of S , there exists a vertex of ∂R in D .*

Proof. Connectivity of $H(S)$ assures that t is not an interior point of R , and there is an edge $e = (u, w)$ of ∂R intersecting vt . If there is a no vertex of ∂R in D , the edge e cuts $D \cup \Delta$ into two connected components as seen in Figure 3. Since e is an edge of $H(S)$, either $d(u, v) \geq d(u, w)$ or $d(w, v) \geq d(u, w)$ (otherwise the lens corresponding to e contains v contradicting the definition of $H(S)$). We can assume that the former case occurs, and hence u is in the same side of w with

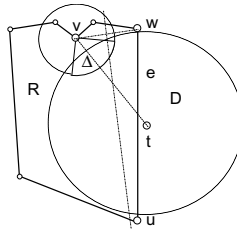


Fig. 3. Proof of Lemma 1

respect to the perpendicular bisector of vw . The segment uw and the center t of D must lie in the same side of the bisector. Also, from the above fact and $w \notin \Delta \cup D$, we have $\angle wvt > \pi/4$. Thus, by elementary geometry, we can see that $d(u, w)$ is greater than the radius of D . Thus, the length of e is greater than 1, contradicting the definition of $H(S)$.

We slightly modify the detour-finding strategy to utilize the above lemma effectively: If the directional routing halts, we first examine emptiness of Δ using a neighbor-find operation. If Δ has a point q of S , we send the packet to q without calling the one-face routing, and the distance is decreased similarly to the usual step of directional routing, since the angle $\angle qvt$ is less than $\pi/4$, and hence less than $\pi/3$. If Δ has no point of S , we can set $\delta = 1/2$, and the one-face routing always finds a point q to decrease the distance to the target t by at least $1/2$. Thus, we have the following:

Theorem 2. *By the above modification, we can reduce the number of calls of one-face routing to $O(d(s, t))$.*

4 Analysis of Number of Hops

Now, let us consider the number of hops on a route. In the following, we ignore the hops required in the detour-finding routine. Clearly, $d(s, t)$ hops are necessary for any routing method. If S is uniformly distributed, the greedy routing on $\text{UDG}(S)$ uses $O(d(s, t))$ hops, which is the best possible. However, greedy routing on $\text{UDG}(S)$ has several defects discussed in the introduction. The Gabriel graph is considered in GOAFR^+ [8], but theoretical asymptotic bound is not known for those methods for a general S . We begin with the following rather weak bound for the general case (proof is routine, and omitted).

Lemma 2. *For any choice of F , the number of hops in the greedy routing for sending a packet from s to t is $O(d(s, t)\alpha^{-1})$.*

The $O(\alpha^{-1}d(s, t))$ bound is tight for $\text{LNG}(S)$ and the Gabriel graph even for a uniform distribution. $\text{RLNG}(S)$ has generally longer edges than $\text{LNG}(S)$, and thus advantageous in number of hops as shown below:

Theorem 3. *For any point set S with the distance condition and for any pair $s, t \in S$, the expected number of hops in the directional routing on $\text{RLNG}(S)$ is $O(d(s, t) \log^2 \alpha^{-1} + \alpha^{-1})$.*

Proof. We only give an intuitive outline, and detailed proof will be given in a full version of this paper. Suppose that we sort the points in $Q(v, k)$ with respect to the distance towards v . If we randomly pick w , it is located in the middle in the sorted list. Now, we can observe $Q(w, k) \cap D(v) \subseteq Q(v, k)$, and thus we consume all the points of $Q(v, k)$ within $O(\log N)$ hops if there are N points. This argument cheats a little, since it is not always true that the target t is located in $Q(w, k)$. But we can prove that we need to consider at most two directions (k and $k + 1$, or k and $k - 1$) while we remains in $D(v)$. This gives $O(\log^2 N)$ bound, and since $N = O(\alpha^{-2})$, we have the theorem.

There is an instance such that the expected number of hops becomes $\Omega(d(s, t) \log \alpha^{-1})$. If S is uniformly distributed and $f_k(v)$ is randomly chosen, the bound of the expected number of hops becomes $O(d(s, t))$, since we can expect a constant improvement of the distance in each step.

5 Dynamic Routing Method of a Mobile Ad Hoc Network

We discuss dynamic maintenance of our routing algorithm for a mobile ad hoc networks. The nodes move freely (even jump discontinuously or appear/disappear) and we cannot anticipate the movement. Thus, the information of movement of a node u is given to another node v only by sending a message from u . However, we assume that the update/movement during sending a packet can be ignored, and also a packet sent to the initially known position of t can reach the current position of t in case we allow t to move.

We first discuss how to maintain $\text{GLNG}(S)$, whose structure is stored at the nodes in the distributed fashion; in other words, we store $f_k(v)$ for $1 \leq k \leq 6$ in each v . There are two possible cases for which want to update the local neighbor: (1). Suppose that w is stored as $f_k(v)$ at v , and it moves by a vector x from the original position w_0 when it was stored at v . If $w_0 + x \in Q(v, k)$, we need not change it. However, if $w_0 + x \notin Q(v, k)$, we need to update $f_k(v)$. Note that v may also move. (2). Suppose that $f_k(v) = *$ and a point w comes into $Q(v, k)$. Then, we want to set $f_k(v) = w$.

We adopt the following *lazy update* strategy. We do not update $f_k(v)$ until v receives a packet that is sent to a destination t that is in $P(v, k)$. The node v keeps a copy of the packet, and tries to send the packet to w stored as $f_k(v)$, together with the information of $Q(v, k)$. Then, w receives the packet and sends back a “success” acknowledgement only if it is located in $Q(v, k)$. Otherwise it responds “no”. Of course, if $w \notin D(v)$, v receives no acknowledgement. If v receives “success”, it discards the copy of the packet; otherwise (including the case where $w = *$), it issues the neighbor-find operation, updates its local neighbors, and re-sends the packet. We note that such a simple update method is impossible for more rigid structures such as Gabriel graphs, Delauney triangulations and $\text{LNG}(S)$, since the existence of an edge (u, w) is affected in those graphs by the location of other nodes.

Unfortunately, it is expensive to explicitly maintain the structure of $H(S)$ in a dynamic situation, and it is not advantageous to spend much work to maintain $H(S)$ if the update is frequent, since the one-face routing is seldom called in a practical situation. Thus, we propose the following *on-the-fly* method for the one-face routing without explicitly updating $H(S)$. The node v issue a neighbor-find operation within the unit disk $D(v)$. We find all edges of $H(S)$ which is known to be in H by only using local information of $D(v)$. All the edges adjacent to v in $H(S)$ are among them, since the lens for each such edge is in $D(v)$. Thus, we find a subchain $C(v) \ni v$ of ∂R . If there is no exit point on $C(v)$, we send two messengers to the ends of the subchain and issue neighbor-find operations there

to extend $C(v)$ until one of them finds an exit point. Once one of the messengers finds an exit point, it comes back to v and we send the packet to the exit point. Here, we temporary record neighbors in $C(v)$ of each node $w \in C(v)$, and assume that $C(v) \subset \text{UDG}(S)$ during the one-face-routing.

6 Experimental Results

Because of space limitation, we only give brief summary of our experimental results. We compare the following four routing strategies: UDG (greedy routing on UDG using neighbor-find operations to find the next hops), GG (greedy routing on the Gabriel graph), LNG (directional routing on LNG), and RLNG (directional routing on RLNG). All simulations are done where 1000 nodes are randomly distributed in a square grid of size 1000×1000 , and packets are sent between random pairs of nodes.

Figure 4 shows the average number of hops in greedy routings. The number of hops of RLNG is only slightly worse than that of UDG, while GG and LNG need more than twice many hops. In figure 5, we give the number of transmissions, including both greedy and face routing routines. Since UDG issues many neighbor-find operations, it needs a lot of transmissions. Thus, we can conclude that RLNG is best in this experiment, and we can enjoy its sparseness and reduction of message transmissions without losing its efficiency in number of hops. We also did experiment for the dynamic case where we adopt the lazy update and on-the-fly one-face routing for RLNG. We experimentally confirmed that RLNG performs best for the dynamic case, too (we omit details in this version).

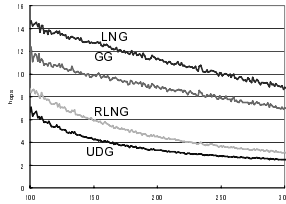


Fig. 4. Number of hops

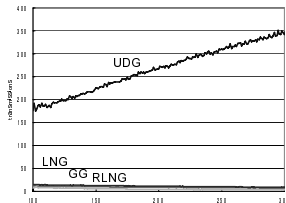


Fig. 5. Transmissions

References

1. Aurenhammer, F.: Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Comput. Surv.* 23(3), 345–405 (1991)
2. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks* 7(6), 609–616 (2001)
3. Calinescu, G., Mandoiu, I.I., Wan, P.-J., Zelikovsky, A.: Selecting forwarding neighbors in wireless ad hoc networks. *MONET* 9(2), 101–111 (2004)
4. Halldórsson, M.M., Tokuyama, T.: Minimizing interference of a wireless ad-hoc network in a plane. *Theor. Comput. Sci.* 402, 29–42 (2008)
5. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: *Mobile Computing*, p. 353 (1996)
6. Katoh, N., Tokuyama, T., Iwano, K.: On minimum and maximum spanning trees of linearly moving points. *Discrete & Computational Geometry* 13, 161–176 (1995)
7. Kranakis, E., Singh, H., Urrutia, J.: Compass routing on geometric networks. In: *Proc. 11th Canadian Conference on Comput. Geom.* (1999)
8. Kuhn, F., Wattenhofer, R., Zhang, Y., Zollinger, A.: Geometric ad-hoc routing: of theory and practice. In: *Proc. 22nd ACM Symposium on Principles of Distributed Computing*, pp. 63–72 (2003)
9. Kuhn, F., Wattenhofer, R., Zollinger, A.: Asymptotically optimal geometric mobile ad-hoc routing. In: *Proc. 6th Int. Workshop, DIAL-M 2002*, pp. 24–33 (2002)
10. Plesse, T., Adjih, C., Minet, P., Laouiti, A., Plakoo, A., Badel, M., Mühlethaler, P., Jacquet, P., Lecomte, J.: OLSR performance measurement in a military mobile ad hoc network. *Ad Hoc Networks* 3(5), 575–588 (2005)
11. Preparata, F.P., Shamos, M.I.: *Computational Geometry - An Introduction*. Springer, Heidelberg (1985)
12. Yao, A.C.-C.: On constructing minimum spanning trees in k -dimensional spaces and related problems. *SIAM J. Comput.*, 721–736 (1982)

Author Index

- Ackerman, Margareta 178
Al-Jubeh, Marwan 192
Alfeld, Chris 7
- Bachmaier, Christian 66
Barford, Paul 7
Bera, Debajyoti 418
Berenbrink, Petra 449
Berman, Kenneth A. 368
Bezáková, Ivona 307
Bhatnagar, Nayantara 307
Bhattacharya, Binay 225
Biedl, Therese 86
Bilò, Vittorio 17
Böcker, Sebastian 258, 277, 297
Brandenburg, Franz J. 66
Brunner, Wolfgang 66
Bui, Quang B.A. 297
- Cai, Jin-Yi 7
Chan, Agnes 148
Chan, Joseph Wun-Tat 358
Chin, Francis Y.L. 358
Cohen, Nathann 37
- Dantchev, Stefan 459
Di Crescenzo, Giovanni 127
Diehl, Scott 429
- Eğecioğlu, Ömer 408
Elbassioni, Khaled 496
- Fagnot, Isabelle 378
Fellows, Michael R. 516
Feng, Wangsen 439
Fenner, Stephen 418
Fertin, Guillaume 378
Fischer, Bernd 287
Flammini, Michele 17
Fomin, Fedor V. 37
Fouz, Mahmoud 158
Friedetzky, Tom 459
Fu, Bin 486
Fujimaki, Ryo 47
Fülöp, Raymund 66
- Green, Frederic 418
Guo, Jiong 516
Gupta, Arvind 268
Gutin, Gregory 37
Gutwenger, Carsten 249
- Heggernes, Pinar 398
Hoffmann, Michael 192
Homer, Steve 418
Hon, Wing-Kai 96
Hu, Yuzhuang 225
Huang, Qiong 138
- Ibarra, Oscar H. 408
Inoue, Youhei 47
Irving, Robert W. 506
Ishaque, Mashhood 192
Izumi, Taisuke 56
Izumi, Tomoko 56
- Kanj, Iyad A. 388
Kao, Ming-Yang 205
Karakostas, George 238
Karpinski, Marek 215
Kehr, Birte 258
Kijima, Shuji 317, 328
Kim, Eun Jung 37
Kiyomi, Masashi 106
Kolliopoulos, Stavros G. 238
Komusiewicz, Christian 516
Kowalczyk, Michael 472
Kratsch, Dieter 388
Kufleitner, Manfred 158
- Lee, Chia-Jung 338
Li, Angsheng 486
Li, Xiang-Yang 439
Lu, Chi-Jen 338
- Mäkinen, Erkki 178
Makino, Kazuhisa 496
Mancini, Federico 398
Manthey, Bodo 158
Matsui, Yasuko 328
Mañuch, Ján 268

- McDermid, Eric 506
 Mchedlidze, Tamara 76
 Min, Kerui 205
 Monaco, Gianpiero 17
 Moscardelli, Luca 17
 Mu, Yi 138
 Muthukrishnan, S. 1
 Mutzel, Petra 249

 Nagel, Lars 459
 Nekrich, Yakov 215
 Nemoto, Toshio 317
 Niedermeier, Rolf 516

 Ono, Hirotaka 56

 Papadopoulos, Charis 398
 Patterson, Murray 268
 Pervukhin, Anton 277
 Peserico, Enoch 348
 Pretto, Luca 348

 Rajaraman, Rajmohan 148
 Randall, Dana 307
 Rasche, Florian 258
 Rauf, Imran 496
 Rudra, Atri 27

 Saitoh, Toshiki 106
 Sato, Kazushige 527
 Sauerwald, Thomas 449
 Saurabh, Saket 37
 Seeber, Patrick 297
 Shi, Qiaosheng 225
 Shioura, Akiyoshi 116
 Simroth, Axel 168
 Souvaine, Diane L. 192
 Souza, Alexander 168
 Šrámek, Rastislav 287
 Sritharan, R. 398
 Stern, Michal 86

 Sun, Zhifeng 148
 Susilo, Willy 138
 Symvonis, Antonios 76

 Takahashi, Toshihiko 47
 Ting, Hing-Fung 358
 Tokuyama, Takeshi 527
 Tóth, Csaba D. 192
 Truss, Anke 297
 Tsai, Shi-Chun 338

 Uehara, Ryuhei 106
 Uhlmann, Johannes 516

 van Melkebeek, Dieter 429
 Vialette, Stéphane 378
 Vicari, Elias 287

 Wada, Koichi 56
 Wang, Biing-Feng 96
 Wang, Jing 238
 Wang, Yajun 439
 Widmayer, Peter 287
 Williams, Ryan 429
 Wong, Duncan S. 138

 Yagiura, Mutsunori 116
 Yamamoto, Masaki 328
 Yang, Guomin 138
 Yegneswaran, Vinod 7
 Yeo, Anders 37
 Yoshikawa, Chad 368
 Yu, Chih-Chiang 96
 Yuen, Tsz Hon 138

 Zeini Jahromi, Nima 158
 Zey, Bernd 249
 Zhang, Liyu 486
 Zhang, Yong 358
 Zhu, Feng 148
 Zhu, Hong 205