Lecture Notes in Computer Science 2108
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Jie Wang (Ed.)

Computing and Combinatorics

7th Annual International Conference, COCOON 2001
Guilin, China, August 20-23, 2001
Proceedings

Springer

Series Editors

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

Volume Editor

Jie Wang
University of Massachusetts, Department of Computer Science
One University Avenue, Lowell, MA 01854, USA
E-mail: wang@cs.uml.edu

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Computing and combinatorics : 7th annual international conference ;
proceedings / COCOON 2001, Guilin, China, August 20 - 23, 2001. Jie Wang
(ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ;
Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2108)
ISBN 3-540-42494-6

CR Subject Classification (1998): F.2, G.2.1-2, I.3.5, C.2.3-4, E.1

ISSN 0302-9743
ISBN 3-540-42494-6 Springer-Verlag Berlin Heidelberg New York

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

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

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH
Printed on acid-free paper SPIN: 10839558 06/3142 5 4 3 2 1 0

Preface

The papers in this volume were presented at the *Seventh Annual International Computing and Combinatorics Conference* (COCOON 2001), held August 20-23, 2001, in Guilin, China. The topics cover most aspects of theoretical computer science and combinatorics related to computing. The conference was held in cooperation with the Chinese Academy of Sciences and China Computer Federation.

Submissions to the conference this year were conducted entirely online. A total of 97 papers were submitted in time to be considered, of which 50 regular papers and 16 short papers were selected by an international program committee consisting of Eric Allender, Bernard Chazelle, Danny Chen, Jianer Chen, Francis Chin, Kyung-Yong Chwa, Rod Downey, Erich Grädel, Yuri Gurevich, Steven Homer, Toshihide Ibaraki, Tao Jiang, Ker-I Ko, D.T. Lee, Xuemin Lin, Maurice Nivat, R. Ravi, Rüdiger Reischuk, Seinosuke Toda, Jie Wang, Lusheng Wang, Guoliang Xue, and Mihalis Yannakakis.

The authors of submitted papers come from the following countries and regions: Australia, Austria, Bangladesh, Canada, China (including Hong Kong and Taiwan), Czech Republic, France, Germany, India, Israel, Italy, Japan, Korea, New Zealand, The Netherlands, Poland, Russia, Singapore, Spain, Switzerland, U.K., and U.S.A. Each paper was given to at least three Program Committee members, who in some cases were assisted by subreferees. In addition to the selected papers, the conference also included two invited presentations by Bernard Chazelle and Avi Wigderson.

To promote young researchers, the Hao Wang Award this year was given to a paper selected from papers written solely by authors who, at the time of submission, were either students or had received their doctoral degrees within the previous five years. I am happy to announce that the recipient of this award was Xiang-Yang Li for his paper "Generating Well-Shaped d-Dimensional Delaunay Meshes".

I would like to thank the program committee co-chair Yuri Gurevich for his strong support and all program committee members, their support staff, and subreferees for their excellent work within demanding time constraints. I would also like to thank all authors for submitting their papers to the conference. I am grateful to Steve Tate for letting me use the ACM SIGACT electronic submission service, Minghui Li for helping me create conference web pages, and Richard Cheek for providing system support. Finally, I would like to express my gratitude to Ding-Zhu Du, Xudong Hu, and all local organizers for their hard work in making this meeting possible and enjoyable.

August 2001 Jie Wang

Program Committee Chairs:
Yuri Gurevich, Microsoft Research, USA
Jie Wang, University of Massachusetts Lowell, USA

Program Committee Members:
Eric Allender, Rutgers University, USA
Bernard Chazelle, Princeton University, USA
Danny Z. Chen, University of Notre Dame, USA
Jianer Chen, Texas A&M University, USA
Francis Chin, University of Hong Kong, China
Kyung-Yong Chwa, Korea Advanced Institute of Science & Technology, Korea
Rod Downey, Victoria University of Wellington, New Zealand
Erich Grädel, Aachen University of Technology, Germany
Steven Homer, Boston University, USA
Toshihide Ibaraki, Kyoto University, Japan
Tao Jiang, University of California at Riverside, USA
Ker-I Ko, State University of New York at Stony Brook, USA
D.T. Lee, Academy of Science, Taiwan
Xuemin Lin, University of New South Wales, Australia
Maurice Nivat, University of Paris, France
R. Ravi, Carnegie Mellon University, USA
Rüdiger Reischuk, University of Lübeck, Germany
Seinosuke Toda, Nihon University, Japan
Lusheng Wang, City University of Hong Kong, China
Guoliang Xue, University of Vermont, USA
Mihalis Yannakakis, Bell Labs, USA

Referees:

Dorit Aharonov
Klaus Ambos-Spies
Douglas Bridges
Marek Chrobak
Xiaotie Deng
Bjarni Halldorsson
Patrick Healy
John Hine
Yingping Huang
Xiaohua Jia
Ming-Yang Kao
Jochen Konemann
Guojun Li
Xue Li
Yanjun Li
Yifan Li
Weifa Liang
Shuang Luan
Catherine McCartin
Anil Nerode
Ojas Parekh
Chong-Dae Park
Jung-Heum Park
Kunsoo Park
C. K. Poon
John Shepherd
Amitabh Sinha
S. Skiena
Michael Trick
Klaus Wagner
C.A. Wang
Wenping Wang
Xiaodong Wu
Baogang Xu
Yuansheng Yang
Dongmo Zhang

Table of Contents

Complexity Theory

Complete Problems for Valiant's Class
of qp-Computable Families of Polynomials 1
Markus Bläser

Log-Space Constructible Universal Traversal Sequences
for Cycles of Length $O(n^{4.03})$ 11
Michal Koucký

On Universally Polynomial Context-Free Languages 21
Nicholas Tran

Separating Oblivious and Non-oblivious BPs 28
Kazuo Iwama, Yasuo Okabe, Toshiro Takase

Program Schemes, Queues, the Recursive Spectrum and Zero-One Laws... 39
Iain A. Stewart

Algebraic Properties for P-Selectivity 49
Lane A. Hemaspaandra, Harald Hempel, Arfst Nickelsen

Parallelizability of Some P-Complete Geometric Problems
in the EREW-PRAM 59
Carla Denise Castanho, Wei Chen, Koichi Wada, Akihiro Fujiwara

Computational Biology

Enhanced Sequence Reconstruction with DNA Microarray Application.... 64
Samuel A. Heath, Franco P. Preparata

Non-approximability of Weighted Multiple Sequence Alignment 75
Bodo Siebert

A Greedy Algorithm for Optimal Recombination 86
Shiquan Wu, Xun Gu

Computational Geometry

Generating Well-Shaped d-dimensional Delaunay Meshes..... 91
Xiang-Yang Li

Towards Compatible Triangulations 101
*Oswin Aichholzer, Franz Aurenhammer, Hannes Krasser,
Ferran Hurtado*

An Improved Upper Bound on the Size of Planar Convex-Hulls 111
Abdullah N. Arslan, Ömer Eğecioğlu

On the Planar Two-Watchtower Problem 121
Sergei Bespamyatnikh, Zhixiang Chen, Kanliang Wang, Binhai Zhu

Efficient Generation of Triconnected Plane Triangulations 131
Shin-ichi Nakano

Packing Two Disks into a Polygonal Environment 142
Prosenjit Bose, Pat Morin, Antoine Vigneron

Maximum Red/Blue Interval Matching with Applications 150
Danny Z. Chen, Xiaobo (Sharon) Hu, Xiaodong Wu

Computing Farthest Neighbors on a Convex Polytope 159
Otfried Cheong, Chan-Su Shin, Antoine Vigneron

Finding an Optimal Bridge between Two Polygons 170
Xuehou Tan

How Good Is Sink Insertion? 181
Xiang-Yang Li, Yu Wang

Polynomial Time Algorithms for Three-Label Point Labeling 191
Rob Duncan, Jianbo Qian, Binhai Zhu

Approximation Algorithms for the Watchman Route and Zookeeper's Problems 201
Xuehou Tan

Data Structures and Algorithms

PC-Trees vs. *PQ*-Trees 207
Wen-Lian Hsu

Stacks versus Deques 218
Holger Petersen

Optimizing a Computational Method for Length Lower Bounds for Reflecting Sequences 228
H.K. Dai

Games and Combinatorics

Competitive Facility Location along a Highway 237
Hee-Kap Ahn, Siu-Wing Cheng, Otfried Cheong, Mordecai Golin, René van Oostrum

Membership for Core of LP Games and Other Games 247
Qizhi Fang, Shanfeng Zhu, Maocheng Cai, Xiaotie Deng

Strong Solutions to the Identification Problem 257
Pino Caballero-Gil, Candelaria Hernández-Goya

Area Efficient Exponentiation
Using Modular Multiplier/Squarer in GF(2^m) 262
Hyun-Sung Kim, Kee-Young Yoo

Graph Algorithms and Complexity

A Space Saving Trick for Directed Dynamic Transitive Closure
and Shortest Path Algorithms 268
Valerie King, Mikkel Thorup

Finding the Most Vital Node of a Shortest Path 278
Enrico Nardelli, Guido Proietti, Peter Widmayer

Algorithm for the Cost Edge-Coloring of Trees 288
Xiao Zhou, Takao Nishizeki

Counting H-Colorings of Partial k-Trees 298
Josep Díaz, Maria Serna, Dimitrios M. Thilikos

A Linear Time Algorithm for Enumerating All the Minimum and Minimal
Separators of a Chordal Graph 308
L. Sunil Chandran

Graph Separators: A Parameterized View 318
Jochen Alber, Henning Fernau, Rolf Niedermeier

On Assigning Prefix Free Codes to the Vertices of a Graph 328
N.S. Narayanaswamy, C.E. Veni Madhavan

A New Measure of Edit Distance Between Labeled Trees 338
Chin Lung Lu, Zheng-Yao Su, Chuan Yi Tang

A Highly Efficient Algorithm to Determine Bicritical Graphs 349
Dingjun Lou, Ning Zhong

Graph Drawing

Layered Drawings of Graphs with Crossing Constraints 357
Irene Finocchi

On the Validity of Hierarchical Decompositions 368
Irene Finocchi, Rossella Petreschi

Graph Theory

Lower Bounds on the Minus Domination and k-Subdomination Numbers . . 375
Liying Kang, Hong Qiao, Erfang Shan, Ding-Zhu Du

Edge Connectivity vs Vertex Connectivity in Chordal Graphs 384
L. Sunil Chandran

Changing the Diameter of Graph Products 390
Ting-Yi Sung, Jeng-Jung Wang

Plane Graphs with Acyclic Complex 395
Baogang Xu

On the Domination Numbers of Generalized de Bruijn Digraphs
and Generalized Kautz Digraphs 400
Yosuke Kikuchi, Yukio Shibata

A Notion of Cross-Perfect Bipartite Graphs 409
Milind Dawande

Some Results on Orthogonal Factorizations 414
Haodi Feng

Cluttered Orderings for the Complete Graph 420
Myra B. Cohen, Charles J. Colbourn, Dalibor Froncek

Online Algorithms

Improved On-Line Stream Merging:
From a Restricted to a General Setting 432
Wun-Tat Chan, Tak-Wah Lam, Hing-Fung Ting, Wai-Ha Wong

On-Line Deadline Scheduling on Multiple Resources 443
Jae-Hoon Kim, Kyung-Yong Chwa

Competitive Online Scheduling with Level of Service 453
Ee-Chien Chang, Chee Yap

On-Line Variable Sized Covering 463
Leah Epstein

Randomized and Average-Case Algorithms

On Testing for Zero Polynomials by a Set of Points
with Bounded Precision 473
Jin-Yi Cai, Eric Bach

A Randomized Algorithm for Gossiping in Radio Networks 483
Marek Chrobak, Leszek Gąsieniec, Wojciech Rytter

Deterministic Application of Grover's Quantum Search Algorithm 493
Kyoichi Okamoto, Osamu Watanabe

Random Instance Generation for MAX 3SAT 502
Mitsuo Motoki

Steiner Trees

The Euclidean Bottleneck Steiner Tree and Steiner Tree with Minimum Number of Steiner Points 509
Dingzhu Du, Lusheng Wang, Baogang Xu

An FPTAS for Weight-Constrained Steiner Trees in Series-Parallel Graphs 519
Guangting Chen, Guoliang Xue

Systems Algorithms and Modeling

Decidable Approximations on Generalized and Parameterized Discrete Timed Automata 529
Zhe Dang, Oscar H. Ibarra, Richard A. Kemmerer

Multiplicative Adaptive Algorithms for User Preference Retrieval 540
Zhixiang Chen

Parametric Scheduling for Network Constraints 550
K. Subramani

A Logical Framework for Knowledge Sharing in Multi-agent Systems 561
Kaile Su, Xudong Luo, Huaiqing Wang, Chengqi Zhang, Shichao Zhang, Qingfeng Chen

A Lockout Avoidance Algorithm without Using Time-Stamps for the k-Exclusion Problem 571
Kumiko Obokata, Michiko Omori, Kazuhiro Motegi, Yoshihide Igarashi

Computability

Prefix-Free Languages and Initial Segments of Computably Enumerable Degrees 576
Guohua Wu

Weakly Computable Real Numbers and Total Computable Real Functions 586
Robert Rettinger, Xizhong Zheng, Romain Gengler, Burchard von Braunmühl

Turing Computability of a Nonlinear Schrödinger Propagator 596
Klaus Weihrauch, Ning Zhong

Author Index 601

Complete Problems for Valiant's Class of qp-Computable Families of Polynomials

Markus Bläser

Institut für Theoretische Informatik, Med. Universität zu Lübeck
Wallstr. 40, 23560 Lübeck, Germany
blaeser@tcs.mu-luebeck.de

Abstract. We prove that the families *matrix powering*, *iterated matrix product*, and *adjoint matrix* are VQP–complete, where VQP denotes Valiant's class of quasipolynomial–computable families of multivariate polynomials. This proves a conjecture by Bürgisser [3, Conjecture 8.1].

1 Introduction

In connection with his famous #P–completeness result [11] for the permanent, Valiant presented an algebraic analogue of the theory of NP–completeness [10] (a decade before the proposal of Blum, Shub, and Smale [2]). In the present work, we prove the completeness of various families of multivariate polynomials for Valiant's algebraic class of qp–computable families of multivariate polynomials. For the reader's convenience, we first present the relevant details of Valiant's theory. More material can be found in [4,3].

A function $t : \mathbb{N} \to \mathbb{N}$ is called *p–bounded* if $t(n) \leq O(n^c)$ for some constant c. The function t is called *p–bounded from above and below* if in addition $t(n) \geq \Omega(n^{1/c})$. A function $u : \mathbb{N} \to \mathbb{N}$ is called *qp–bounded* if $u(n) \leq n^{O(\log^c n)}$.

The main objects in Valiant's algebraic NP–completeness theory are certain families of multivariate polynomials. Throughout the remainder of this work, k denotes a field.

Definition 1. *A sequence $f = (f_n)$ of multivariate polynomials over k is called a p–family if the number of variables and the degree of f_n are both p–bounded functions in n.*

The *complexity* $L(f)$ of a multivariate polynomial $f \in k[X_1, \ldots, X_m]$ is the size of a smallest arithmetic circuit over the basis $\{+, -, *\}$ (all operations with fanin two) that computes f from the indeterminates $X_1, \ldots, X_m$ and the scalars in k.

Definition 2. *1. A p–family $f = (f_n)$ is* p–computable *if the complexity $L(f_n)$ is a p–bounded function in n. The class of all p–computable families is denoted by* VP *(or* VP_k*).*

2. The p–family f is qp–computable *if $L(f_n)$ is a qp–bounded function in n. The class of all qp–computable families is denoted by* VQP *(or* VQP_k*).*

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 1–10, 2001.
© Springer-Verlag Berlin Heidelberg 2001

In the same way, we can also define the class VNP, an algebraic analogue of NP. Since a detailed definition of VNP is more complicated than those of the classes VP and VQP, the reader is referred to [4,3]. The following hypotheses play a central role in Valiant's theory—like Cook's Hypothesis in structural complexity theory.

Hypothesis 1. *1.* Valiant's Hypothesis: $\mathsf{VP} \subsetneq \mathsf{VNP}$ *over any field.*
2. Extended Valiant's Hypothesis: $\mathsf{VNP} \setminus \mathsf{VQP} \neq \emptyset$ *over any field.*

In Valiant's model, families of polynomials are compared by a simple type of reductions, namely *projections*. A polynomial $f(X_1, \ldots, X_m)$ is called a projection of another polynomial $g(Y_1, \ldots, Y_n)$ if there are $s_1, \ldots, s_n \in k \cup \{X_1, \ldots, X_m\}$ such that $f(X_1, \ldots, X_m) = g(s_1, \ldots, s_n)$. In other words, f is obtained from g by replacing each of the $Y_1, \ldots, Y_n$ by a scalar from k or by one of the $X_1, \ldots, X_m$.

Definition 3. *1. A family $f = (f_n)$ is a p–projection of $g = (g_m)$, in symbols $f \leq_{\mathrm{p}} g$, if there exists a function r that is p–bounded from above and below such that*

$$\exists n_0 \, \forall n \geq n_0 : \; f_n \text{ is a projection of } g_{r(n)} \, .$$

2. A family $f = (f_n)$ is a qp–projection of $g = (g_m)$, in symbols $f \leq_{\mathrm{qp}} g$, if there exists a qp–bounded function r that is also p–bounded from below such that

$$\exists n_0 \, \forall n \geq n_0 : \; f_n \text{ is a projection of } g_{r(n)} \, .$$

The relations "is a p–projection of" and "is a qp–projection of" are both transitive. Obviously, if $g \leq_{\mathrm{p}} f$, then also $g \leq_{\mathrm{qp}} f$.

The next step is to define VQP–complete families.

Definition 4. *A qp–computable family f is called* VQP*–complete if $g \leq_{\mathrm{qp}} f$ for all $g \in \mathsf{VQP}$.*

As expected, if f is VQP–complete and $f \leq_{\mathrm{qp}} h$, then h is also VQP–complete.

Up to now, the only known VQP–complete family is the *determinant* family defined by

$$DET_n = \det(X_{i,j})_{1 \leq i,j \leq n} \, .$$

(For a proof, see [3, Corollary 2.29].) As our main results, we exhibit further VQP–complete families of multivariate polynomials. Specifically, we show that the families *iterated matrix multiplication IMM*, *matrix powering POW*, and *adjoint matrix ADJ* defined by

$$\begin{aligned} IMM_n &= \operatorname{trace}(X_1 \cdots X_n), \\ POW_n &= \operatorname{trace}(X^{\lfloor n/3 \rfloor}), \\ ADJ_n &= \operatorname{trace}(\det(X) \cdot X^{-1}) \end{aligned}$$

are all VQP–complete. (Above, X and the X_ν denote $n \times n$–matrices with independent indeterminate entries.) This proves a conjecture by Bürgisser [3, Conjecture 8.1]. The original conjecture for POW made by Bürgisser is that $(\operatorname{trace}(X^n))$ is VQP–complete. This remains an open problem.

The main technical problems we are confronted with are that we have a very weak type of reductions, namely qp–projections, and that we only compute one result.

For any constant $c \in \mathbb{N}$, let the family *constant size c iterated matrix multiplication c–CIMM* be defined through

$$c\text{–}CIMM_n = \operatorname{trace}(Y_1 \cdots Y_n)\,,$$

where the Y_ν are $c \times c$–matrices (and *not* $n \times n$–matrices) with independent indeterminate entries. Strengthening the above VQP–completeness result for *IMM*, we also show that for any $c \geq 3$, the family *c–CIMM* is VQP–complete. This follows from the results by Ben-Or and Cleve [1].

2 Auxiliary Results

We briefly review some (well known) results, which we will need later on, and provide some useful variations of these auxiliaries. For a multivariate polynomial $f \in k[X_1, \ldots, X_m]$ the *depth* $D(f)$ is the minimal depth of an arithmetic circuit over the basis $\{+, -, *\}$ (all operations with fanin two) that computes f from the indeterminates $X_1, \ldots, X_m$ and the scalars in k.

Theorem 1 (Hyafil, Valiant et al.). *If f is an n–variate polynomial of degree $d \geq 1$, then*

$$D(f) \leq O(\log(d \cdot L(f)) \log d + \log n).$$

Moreover, there is an arithmetic circuit of depth $O(\log(d \cdot L(f)) \log d + \log n)$ and size $d^6 L(f)^3$ that computes f.

For a proof of the first part of the statement, see [7]. It can also be shown that the constant hidden in the O–notation is universal. The second claim is due to Valiant et al. [9]. The reader is also referred to [4, Thm. 21.35]. Miller et al. [8] present an online construction.

For technical reasons, we introduce the family *c–CIMM′* defined through

$$c\text{–}CIMM'_n = \text{the entry in position } (1,1) \text{ of } Y_1 \cdots Y_n,$$

where the Y_ν are $c \times c$–matrices with independent indeterminate entries. The family 3–*CIMM′* plays an important role for our completeness results.

Theorem 2 (Ben-Or & Cleve). *Let f be a multivariate polynomial. Moreover, let $\ell = 4^{D(f)}$. Then f is a projection of 3–$CIMM'_\ell$.*

For a proof, see [1, Thm. 1]. Since Ben-Or and Cleve focused on computing polynomial size formulas with polynomial length straight-line programs that use a constant number of registers, they only stated the above theorem for formulas. Nevertheless, Ben-Or and Cleves approach also works for arbitrary straight-line programs, since a straight-line program of depth d induces a formula of depth d and size at most 2^d.

Since

$$(1,1)\text{–entry of } Y_1 \cdots Y_n = \operatorname{trace}(E_{1,1} \cdot Y_1 \cdots Y_n \cdot E_{1,1}),$$

where $E_{1,1}$ denotes the $c \times c$–matrix that has a one in position $(1,1)$ and zeros elsewhere, $c\text{–}CIMM'_n$ is a projection of $c\text{–}CIMM_{n+2}$. Therefore, the below theorem follows.

Theorem 3. *Let f be a multivariate polynomial and let $\ell = 4^{D(f)} + 2$. Then f is a projection of $3\text{–}CIMM_\ell$.*

3 Completeness Results

In this section, we prove as our main result the VQP–completeness of the families *IMM*, *POW*, and *ADJ*. Since each of these three families is in VP, it suffices to show the hardness.

3.1 Iterated Matrix Multiplication

Theorem 4. *For any $c \geq 3$, c–CIMM is VQP–complete (under qp–projections). The same is true for c–CIMM′.*

Proof. The polynomial $c\text{–}CIMM_n$ can be computed by an arithmetic circuit of size $O(n)$. Thus $c\text{–}CIMM \in \mathsf{VP} \subseteq \mathsf{VQP}$.

Since obviously $c\text{–}CIMM_n$ is a projection of $c'\text{–}CIMM_n$ for each n and each $c' \geq c$, it is sufficient to show that for any $f = (f_n) \in \mathsf{VQP}$, $f \leq_{\mathrm{qp}} 3\text{–}CIMM$. By definition, we know that the number of variables and the degree of f_n are p–bounded functions and that $L(f_n)$ is a qp-bounded function. By Theorem 1, $D(f_n) \leq O(\log^m n)$ for some constant m. By Theorem 3, f_n is a projection of $3\text{–}CIMM_\ell$ where $\ell = n^{O(\log^{m-1} n)}$. Hence f is a qp–projection of $3\text{–}CIMM$.

The result for $c\text{–}CIMM'$ follows completely alike. □

Since $3\text{–}CIMM_n$ is a projection of IMM_n, we immediately obtain the following corollary.

Corollary 1. *The family IMM is VQP–complete (under qp–projections).*

The next corollary is of rather technical nature. It describes a "self reducibility" property of $c\text{–}CIMM$. We will rely on the corollary in the subsequent sections.

Corollary 2. *Let $c \geq 3$ and let i and j be functions $\mathbb{N} \to \{1, 2, \ldots, c\}$. Furthermore, let (α_n) and (β_n) be sequences of scalars from k and define $w(n) = \min\{i \geq n \mid \alpha_i \neq 0\}$. If w is qp–bounded, then $(\alpha_n \cdot f_n + \beta_n)$ is VQP–complete, where f_n is the entry in position $(i(n), j(n))$ of a generic iterated 3×3–matrix product of length n.*

Proof. We first prove that $(c\text{–}CIMM'_n + \beta_n)$ is VQP–complete. Then we reduce $(c\text{–}CIMM'_n + \beta_n)$ to $(\alpha_n \cdot f_n + \beta_n)$.

For the first part, let $(g_n) \in \mathsf{VQP}$ be given. Then also $(g_n - \beta_n) \in \mathsf{VQP}$. By the VQP–completeness of $c\text{–}CIMM'$, $(g_n - \beta_n)$ is a qp–projection of $c\text{–}CIMM'$. Thus (g_n) is a qp–projection of $(c\text{–}CIMM'_n + \beta_n)$. Since (g_n) was arbitrary, this shows the VQP–completeness of $(c\text{–}CIMM'_n + \beta_n)$.

Now we reduce $(c\text{–}CIMM'_n + \beta_n)$ to $(\alpha_n \cdot f_n + \beta_n)$. To obtain $c\text{–}CIMM'_n + \beta_n$, we substitute $w(n+3) - n - 3$ many copies of the $c \times c$–identity matrix and one $\alpha^{-1}_{w(n+3)}$ multiple of the $c \times c$–identity matrix into $\alpha_{w(n+3)} \cdot f_{w(n+3)} + \beta_{w(n+3)}$ at arbitrary places except the first and the last matrix. (Note that $w(\nu) \geq \nu$ for all ν.) As the first and the last matrix, we substitute appropriate permutation matrices that move the entry in position $(i(n), j(n))$ into position $(1, 1)$. Since w is qp–bounded, $(c\text{–}CIMM'_n + \beta_n)$ is a qp–projection of $(\alpha_n \cdot f_n + \beta_n)$. □

3.2 Matrix Powering

Our plan is to modify the reduction to iterated matrix multiplication given by Cook [5]. The weak type of reductions however is a major obstacle.

Let $X_1, \ldots, X_m$ be 3×3–matrices with independent indeterminate entries. Let Y be the block matrix of size $3m \times 3m$

$$Y = \begin{pmatrix} 0 & X_1 & 0 & \cdots & 0 \\ 0 & 0 & X_2 & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & X_{m-1} \\ X_m & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

(Note the extra X_m in the lower left corner. Without it, all powers of Y would have trace zero.) A straight forward calculation shows that

$$Y^m = \begin{pmatrix} P_1 & 0 & 0 & \cdots & 0 \\ 0 & P_2 & 0 & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \cdots & P_m \end{pmatrix}, \tag{1}$$

where $P_\mu = X_\mu X_{\mu+1} \cdots X_m X_1 \cdots X_{\mu-1}$. By the below fact,

$$\operatorname{trace}(Y^m) = m \cdot \operatorname{trace}(X_1 \cdots X_m) = m \cdot 3\text{–}CIMM_m.$$

Fact 1. *1. For any two $\ell \times \ell$–matrices A and B,* $\operatorname{trace}(AB) = \operatorname{trace}(BA)$.

2. If we write $A = (A_{i,j})_{1 \leq i,j \leq q}$ as a block matrix with $(\ell/q) \times (\ell/q)$–blocks $A_{i,j}$, then $\operatorname{trace} A = \operatorname{trace} A_{1,1} + \operatorname{trace} A_{2,2} + \cdots + \operatorname{trace} A_{q,q}$.

Thus $m \cdot 3$–$CIMM_m$ is a projection of POW_{3m}. If the characteristic of k is zero, then $m \cdot 3$–$CIMM_m$ is always nonzero. If the characteristic $p > 0$, then at least every second of the $m \cdot 3$–$CIMM_m$ is nonzero. Thus the function $j \mapsto \min\{i \geq j \mid i \not\equiv 0 \ \ (p)\}$ is qp–bounded. Hence by Corollary 2, the family $(m \cdot 3\text{–}CIMM_m)$ is VQP–complete and so is POW.

We define the family m–POW for a given function $m : \mathbb{N} \to \mathbb{N}$ by

$$m\text{–}POW_n = \operatorname{trace}(X^{m(n)}),$$

where X denotes a matrix with independent indeterminate entries of size $n \times n$.

For functions m such that $m(n) \leq \lfloor n/3 \rfloor$ for all n, we can embed the above $\lfloor n/3 \rfloor$–powering problem into the $m(n)$–powering problem by placing the $3m(n) \times 3m(n)$–matrix X together with an $(n - 3m(n)) \times (n - 3m(n))$–zero matrix on the diagonal:

$$Z = \begin{pmatrix} X & 0 \\ 0 & 0 \end{pmatrix}.$$

Thus $\operatorname{trace}(Z^{m(n)}) = \operatorname{trace}(X^{m(n)}) = POW_{3m(n)}$ is a projection of m–POW_n. Therefore, the below theorem follows (by applying the previous construction to $POW_{3m(n)}$, with possibly some of the X_μ substituted by identity matrices).

Theorem 5. *For any function m such that $m(n) \leq \lfloor n/3 \rfloor$ for all n and m is p–bounded from below, the family m–POW is VQP–complete (under qp–projections).*

3.3 Matrix Inversion

Let $X_1, \ldots, X_n$ be 3×3–matrices with indeterminate and independent entries. Let N be the $3(n+1) \times 3(n+1)$–matrix

$$N = \begin{pmatrix} 0 & X_1 & 0 & \cdots & 0 \\ 0 & 0 & X_2 & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & X_n \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}. \tag{2}$$

The matrix N is nilpotent and we have

$$(I - N)^{-1} = I + N + N^2 + \cdots + N^n = \begin{pmatrix} I & P_{1,1} & P_{1,2} & \cdots & P_{1,n} \\ 0 & I & P_{2,2} & & P_{2,n} \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & P_{n,n} \\ 0 & 0 & 0 & \cdots & I \end{pmatrix},$$

where I denotes the identity matrix of appropriate size ($3n+3$ on the left-hand and 3 on the right-hand side) and $P_{i,j}$ stands for the product $X_i \cdots X_j$ for $i \leq j$.

Let S be the permutation matrix

$$S = \begin{pmatrix} 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & I \\ I & 0 & 0 & \cdots & 0 \end{pmatrix}.$$

We have

$$(S^{-1}(I-N))^{-1} = (I-N)^{-1}S = \begin{pmatrix} P_{1,n} & I & P_{1,1} & \cdots & P_{1,n-1} \\ P_{2,n} & 0 & I & & P_{2,n-2} \\ \vdots & & \ddots & \ddots & \vdots \\ P_{n,n} & 0 & 0 & \ddots & I \\ I & 0 & 0 & \cdots & 0 \end{pmatrix}. \tag{3}$$

Thus

$$\operatorname{trace}(S^{-1}(I-N))^{-1} = \operatorname{trace} P_{1,n} = 3\text{–}CIMM_n.$$

Because $\det(S^{-1}(I-N)) = \pm 1$, ± 3–$CIMM_n$ is a projection of $ADJ_{3(n+1)}$. (Since S is a permutation matrix, $S^{-1}(I-N)$ is a projection of the generic $3(n+1) \times 3(n+1)$–matrix X.) This yields the following completeness result.

Theorem 6. *The family ADJ is* VQP*–complete (under qp–projections).*

4 Further Directions

What happens if we do not take the trace in the definition of the families IMM, POW, and ADJ, but look at the following families given by

$$\begin{aligned} IMM'_n &= (1,1)\text{–entry of } X_1 \cdots X_n, \\ POW'_n &= (1,1)\text{–entry of } X^{\lfloor n/3 \rfloor}, \\ ADJ'_n &= (1,1)\text{–entry of } \det(X) \cdot X^{-1}\,? \end{aligned}$$

From Theorem 4, (1), and (3), respectively, it follows that these families are also VQP–complete. Thus we have the following theorem.

Theorem 7. *The above families IMM', POW', and ADJ' are* VQP*–complete (under qp–projections).*

Remark 1. By applying appropriate permutation matrices, we could also define IMM' and ADJ' not by taking the $(1,1)$–entry but any other entry and would still get the same completeness results for these families. Since Valiant's model is nonuniform, the entries may also vary throughout the whole sequence.

Can we achieve a similar result for the family POW'? In fact, we can. The proof is however a little more elaborate.

Theorem 8. *For any functions $i, j : \mathbb{N} \to \mathbb{N}$ with $1 \le i(n), j(n) \le n$ for all n, the family $((i(n), j(n))$–entry of $X^{\lfloor n/3 \rfloor})$ is a VQP–complete family (under qp–projections).*

Proof. Let w.l.o.g. $n = 3m$. We distinguish the cases $i(n) = j(n)$ and $i(n) \neq j(n)$. In the first case, note that in (1), each entry on the diagonal corresponds to an entry of an iterated product of 3×3–matrices of length m.

For the case $i(n) \neq j(n)$, note that also the entry in position $(1, 2)$ of Y^n in (1) is an entry of an iterated 3×3–matrix product. Let P be the permutation matrix that exchanges the first with the $i(n)$th and the second with the $j(n)$th row resp. column. The matrix P is its own inverse. We now consider PYP instead of Y in (1). We have $(PYP)^m = PY^mP$. By the choice of P, the entry in position (i, j) of PY^mP is an entry of an iterated 3×3–matrix product.

Altogether, we obtain a VQP–complete family. □

Over fields of characteristic zero, we can even prove the VQP–completeness of the family $((1, 1)$–entry of $X^n)$ corresponding to the n–powering problem, a task which remains unsolved in the case of taking the trace.

Theorem 9. *Let k be a field of characteristic zero. For any $i, j : \mathbb{N} \to \mathbb{N}$ with $1 \le i(n), j(n) \le n$ for all n and any polynomial p, $((i(n), j(n))$–entry of $X^{p(n)})$ is a VQP–complete family (under qp–projections).*

Proof. Since p is a polynomial, f is in $\mathsf{VP} \subseteq \mathsf{VQP}$.

It remains to show the hardness: let $X_1, \ldots, X_n$ be 3×3–matrices and define N like in (2). The matrix N is nilpotent, specifically, $N^{n+1} = 0$. We have

$$(I+N)^{p(n)} = \sum_{\nu=0}^{n} \binom{p(n)}{\nu} N^\nu = \begin{pmatrix} I & p(n) \cdot P_{1,1} & \binom{p(n)}{2} \cdot P_{1,2} & \cdots & \binom{p(n)}{n} \cdot P_{1,n} \\ 0 & I & p(n) \cdot P_{2,2} & & \binom{p(n)}{n-1} \cdot P_{2,n} \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & p(n) \cdot P_{n,n} \\ 0 & 0 & 0 & \cdots & I \end{pmatrix},$$

where I denotes the identity matrix of appropriate size and $P_{i,j}$ stands for the product $X_i \cdots X_j$.

If $i(n) \neq j(n)$, then we can move the entry in position $(1, n)$ to position $(i(n), j(n))$ by switching over from $(I + N)$ to $P(I + N)P$ for an appropriate permutation matrix (like in the proof of Theorem 8). After that, the entry in position $(i(n), j(n))$ is the $\binom{p(n)}{n}$–multiple of an entry of an iterated 3×3–matrix product.

It remains the case $i(n) = j(n)$. We may assume that $i(n) = j(n) = 1$. Otherwise, we switch over to $P(I + N)P$ where P is the permutation matrix that exchanges the first and the $i(n)$th row resp. column. Let T be the matrix

$$T = \begin{pmatrix} I & 0 & 0 & \cdots & 0 \\ 0 & I & 0 & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 0 \\ I & 0 & 0 & \cdots & I \end{pmatrix}.$$

The inverse of T coincides with T except that the identity matrix in the lower left corner has to be replaced by $-I$. A straight forward calculation shows that the 3×3–matrix in the upper left block of $(T^{-1}YT)^{p(n)} = T^{-1}Y^{p(n)}T$ equals $\binom{p(n)}{n} \cdot P_{1,n} + I$. Thus the entry in position $(1, 1)$ of $(T^{-1}YT)^{p(n)}$ is just $\binom{p(n)}{n} \cdot 3\text{–}CIMM'_n + 1$.

Since the characteristic of k is zero, the coefficient $\binom{p(n)}{n} \neq 0$. Together with Corollary 2 this proves the claim of the theorem. □

Once we have proven the completeness of IMM, POW, and ADJ, many other families of polynomials corresponding to problems from linear algebra turn out to be VQP–complete.

As an example, we examine the family $LINEQ$

$$LINEQ_n = \text{first entry of } \det(X) \cdot X^{-1}b,$$

which corresponds to solving a system of linear equations $Xa = b$. Above, X denotes an $n \times n$–matrix and b a (column) vector of length n with independent indeterminate entries.

If we specialize $b_1 \mapsto 1$ and $b_\nu \mapsto 0$ for $2 \leq \nu \leq n$, then the image of $LINEQ_n$ under this specialization is ADJ'_n. Thus, the family $LINEQ$ is VQP–complete.

Theorem 10. *The family LINEQ is* VQP*–complete (under qp–projections).*

Since the permanent family PER is VNP–complete over fields of characteristic other than two, the Extended Valiant's Hypothesis is equivalent to the following completely algebraic statements by our VQP–completeness results: the Extended Valiant's Hypothesis holds iff PER is not a qp–projection of IMM nor of POW nor of ADJ nor of any other of the VQP–complete families considered in this work.

Acknowledgements

I would like to thank Peter Bürgisser for drawing my attention to the problems considered in the present work. The referees made a lot of thoughtful comments, which helped improving this work.

References

1. Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.
2. L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the Amer. Math. Soc.*, 21:1–46, 1989.
3. Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory.* Springer, 2000.
4. Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory.* Springer, 1997.
5. Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Computation*, 64:2–22, 1985.
6. Joachim von zur Gathen. Feasible arithmetic computations: Valiant's hypothesis. *J. Symbolic Comput.*, 4:137–172, 1987.
7. Laurent Hyafil. On the parallel evaluation of multivariate polynomials. *SIAM J. Comput.*, 8(2):120–123, 1979.
8. Gary L. Miller, Vijaya Ramachandran, and Erich Kaltofen. Efficient parallel evaluation of straight-line code and arithmetic circuits. *SIAM J. Comput.*, 17(4):686–695, 1988.
9. L. G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.
10. Leslie G. Valiant. Completeness classes in algebra. In *Proc. 11th Int. ACM Symp. on Theory of Computing (STOC)*, pages 249–261, 1979.
11. Leslie G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8:189–201, 1979.

Log-Space Constructible Universal Traversal Sequences for Cycles of Length $O(n^{4.03})$
(Extended Abstract)

Michal Koucký*

Rutgers University, Piscataway NJ 08854, USA
mkoucky@paul.rutgers.edu

Abstract. The paper presents a simple construction of polynomial length universal traversal sequences for cycles. These universal traversal sequences are log-space (even NC^1) constructible and are of length $O(n^{4.03})$. Our result improves the previously known upper-bound $O(n^{4.76})$ for log-space constructible universal traversal sequences for cycles.

1 Introduction

One of the major problems in computer science is the graph s,t-connectivity problem. It is well known that a directed version of this problem is complete for nondeterministic log-space (NL), and its undirected version is complete for symmetric log-space (SL). It has been conjectured that symmetric log-space is equal to deterministic log-space ($SL = L$).

One approach to proving that conjecture is to use universal traversal sequences introduced by Cook (see [AKL+].) A traversal sequence for a d-regular graph is a sequence of numbers from $\{1, \ldots, d\}$, which directs us in traversing the graph. For given d and n, a universal traversal sequence for d-regular graphs of size n is a traversal sequence which completely traverses any d-regular graph of size n starting at any vertex. [AKL+] gave a probabilistic argument for existence of universal traversal sequences of polynomial length in d and n. In a sequence of papers [N], [NSW], [ATWZ] it was shown using derandomization of that argument that $SL \subseteq DSPACE(\log^{4/3} n)$.

A more direct approach to proving $SL = L$ is to construct an explicit log-space constructible universal traversal sequence for d-regular graphs, where $d \geq 3$. Indeed, explicit constructions of universal traversal sequences for specific classes of graphs exist. For 2-regular graphs (cycles), [BBK+], and [B] constructed universal traversal sequences of length $n^{O(\log n)}$, which were later superseded by log-space constructible universal traversal sequences for cycles of length $O(n^{4.76})$ by Istrail [I]. In this paper we present a simple log-space construction of universal traversal sequences for cycles of length $O(n^{4.03})$ (which we believe can be improved up to $O(n^{3.99})$ by setting parameters of the construction optimally.) The best known lower bound $\Omega(n^{1.43})$ and non-constructive upper

* Supported in part by NSF grant CCR-9734918.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 11–20, 2001.
© Springer-Verlag Berlin Heidelberg 2001

bound $O(n^3)$ on the length of universal traversal sequences for cycles are due to [BT] and [A], respectively. (See [BT] for comprehensive overview. Recently, it was shown in [K2] that a different kind of traversal sequences does not obey these lower bounds.) Explicit universal traversal sequences of length $n^{O(\log n)}$ for complete graph were presented in [KPS], and [HW] presented construction of length $n^{O(d \log d)}$ for d-regular expanders. The best construction of universal traversal sequences for 3-regular graphs is based on Nisan's pseudo-random generator [N]; these sequences are of length $n^{O(\log n)}$.

1.1 Overview of the Construction

Our construction has two parts:

1. The first part (Section 3) is a construction of an $n^{O(\log n)}$ universal traversal sequence (UTS) for cycles of length n. This part uses essentially the same idea as a construction of Bridgland [B]. It is a recursive construction in which at every iteration a UTS for cycles of length cn, $c \geq 2$, is constructed from a UTS for cycles of length n. Every iteration consists of two dual stages. The depth of the recursion is $O(\log n)$ and the factor by which the generated universal traversal sequence is expanded at each stage is $O(n)$.

2. The second part (Section 4) involves again two dual stages that reduce the expansion factor to $O(1)$, and they interleave the stages of the first part.

At both stages of the first part the traversed graph is reduced to a smaller one, whereas at both stages of the second part the graph is expanded a little bit by inserting new vertices. After applying all four stages the graph is smaller by at least a constant factor.

The main contribution of this paper lies in the construction of the second part where we show that we can modify the traversed graph by inserting new vertices so that this modification is transparent for traversal sequences.

The previous construction by Istrail was also based on Bridgland's construction. Istrail's construction also reduced the expansion factor to a constant as our does but using slightly different (ad hoc) approach. Our construction seems more natural and better motivated hence, much simpler to understand.

2 Preliminaries

Let $G = (V, E)$ be an undirected cycle. Let $V = \{v_1, \ldots, v_n\}$ so that $E = \{(v_i, v_{i+1});\ 1 \leq i < n\} \cup \{(v_1, v_n)\}$. We may look at every edge (u, v) in G as a pair of two directed edges (u, v) and (v, u). For $1 \leq i < n$, the directed edges (v_i, v_{i+1}) and (v_n, v_1) are called the *right edges*, and (v_{i+1}, v_i) and (v_1, v_n) are called the *left edges*. Similarly, v_{i+1} is called a *right neighbor* of v_i, and v_i is a *left neighbor* of v_{i+1}. At every vertex in G, let one of the outgoing edges be labeled by 1 and the other one by 0. We label every vertex in G according to the label of its right outgoing edge.

A 0-1 sequence $b(v_1), b(v_2), \ldots, b(v_n)$ uniquely describes G, where $b(v_i)$ is the label of vertex v_i. We call such a sequence a *graph sequence* corresponding to G.

Two graph sequences are considered to be the same if one of them is a cyclic shift of the other one. Hence, we consider any graph sequence to be wrapped around (cyclic). We consider all the graphs with the same graph sequence to be the same. We identify every graph sequence with its corresponding graph (actually a class of the same graphs.) We refer to the digits of a graph sequence as *vertices*. (We use term vertex also for digits of any 0-1 sequence.)

A *traversal sequence* is a 0-1 sequence. "*To run a traversal sequence t on a graph G starting at a vertex v*" means to walk in G starting at v and following edges labeled consistently with t. Note, if the label of a vertex in the traversal sequence matches the label of the traversed vertex then we move to the right neighbor, otherwise to the left one. Because we identify a graph with its graph sequence we may also talk about running a traversal sequence on a 0-1 sequence.

Let s be a 0-1 sequence. A 0-*run* (1-*run*) in s is any maximal part of s consisting only of 0's (1's). The type of the run, 0-run or 1-run, is called a *color* of that run. The leftmost and rightmost vertices of the run are called *border vertices*. The border vertices of a run of length one coincide. A 0-1 sequence consisting only of runs of length i and j is (i, j)-sequence e.g., 000 111 000 111111 000 is a $(3, 6)$-sequence. A sequence consisting of a single run is called *monochromatic*.

A 0-1 sequence t *completely traverses* s starting at vertex v if during traversal of s by t starting at v we visit all the vertices of s. The 0-1 sequence t *strongly traverses* s starting at v if it completely traverses s starting at v and for every vertex there is a visit to that vertex which comes from left and continues to the right, or it comes from right and continues to the left i.e., the vertex is entered using one edge and left using the other one. A 0-1 sequence t is a *(strong) universal traversal sequence* for cycles of length n if for any cycle G of length n, t completely (strongly) traverses G starting at any vertex.

Observe, any UTS for cycles of length $n+2$ is a UTS for cycles of length $n+1$ and a strong UTS for cycles of length n. (Proofs of many claims are omitted in this extended abstract due to the space limitation; see [K1] for proofs.)

3 An $n^{O(\log n)}$ Universal Traversal Sequence

3.1 A Parity Contraction

Consider running a 1-run t of length l on a 1-run r_1 of length $n \leq l$ starting at a left border vertex of r_1. Assume that r_1 is followed by a 0-run r_0. It is easy to see that if the parity of l and n are the same then we end up at the left border vertex of r_0, else we end up at the right border vertex of r_1. By symmetry, we get symmetric behavior for running a 0-run on a 0-run starting at the left border vertex, and for running a 1-run on a 0-run and a 0-run on a 1-run starting at the right border vertices. Hence, if the parity of the length of the traversed run corresponds to that of the traversal run we end up at the border vertex of the next run, otherwise we end up at the opposite border vertex of the same run.

This motivates the following definition. Let s be a non-monochromatic graph sequence, and let s be the same as $r_0 r_1 \cdots r_{k-1}$, where every r_i is a monochromatic run and for every $0 \leq i < k$, r_i has an opposite color than $r_{(i+1) \bmod k}$.

For $0 \leq i < k$, let r'_i be a run of the same color as r_i, and let the length of r'_i be one if r_i is of odd length and two otherwise. Then a *parity contraction* of s is a sequence $r'_0 r'_1 \cdots r'_{k-1}$. I.e., we replace every run of odd length by one vertex run, and every run of even length by two vertex run.

There is a natural mapping of border vertices of original runs to border vertices of reduced runs. In the case of odd length runs, both border vertices are mapped to the same vertex. Observe that for traversing a parity contraction it suffices to consider (1,2)-sequences.

Proposition 1. *Let s be a non-monochromatic graph sequence of length l, and let $s_\oplus$ be its parity contraction. Let v be a border vertex in s and $v_\oplus$ the corresponding border vertex in $s_\oplus$. If a traversal (1,2)-sequence $t_\oplus$ strongly traverses $s_\oplus$ starting at $v_\oplus$ then t, obtained from $t_\oplus$ by inflating every run by $l + (l \bmod 2)$ vertices, strongly traverses s starting at v.*

3.2 A Pair Contraction

Let s be a (1,2)-sequence. We call a 0-run (1-run) of length 2 in s a 00-*pair* (11-*pair*). A 0-run and a 1-run of length 1 are *singletons* and any maximal part of s consisting of singletons is 01-*run*. (A 01-run does not necessary start by 0 and end by 1.) Let us consider a graph sequence s containing a 01-run followed by a 11-pair and another 01-run: 01010 11 0101010101 ..., and let us consider a traversal sequence $t = 01010101010\ 11\ 01010101\ldots$. Starting t at the first vertex of s, we reach the left vertex v_L of 11-pair after 5 digits of t. The next 1 in t brings us to the right vertex v_R of the 11-pair in s, the following 0 in t takes us back to the vertex v_L, and so on. When 11-pair in t comes we get to the vertex following the 11-pair in s and we continue to run to the right. By symmetry, we observe similar effects for the other possible colorings of the pairs in s and t.

Hence, if s is a graph (1,2)-sequence and we traverse s by a (1,2)-sequence t then whenever we reach a pair in s during that traversal we stay at that pair until a pair appears in t and if the color of the pairs in t and s is the same then after that pair in t, the traversal goes to the right neighbor of the pair in s otherwise to the left one. Even if several pairs are next to each other in s not separated by any 01-run, this behavior of the traversal can be observed on each of these pairs.

This motivates the following construction. Let s be a (1,2)-sequence containing at least one pair. A *pair contraction* s_c of s is a 0-1 sequence obtained from s by first removing all singletons, and then replacing every pair by one vertex of the same color e.g., 01010 11 010 11 01 00 10... is contracted to 110.... Note, the length of s_c is at most half of the length of s.

Proposition 2. *Let s_c be a pair contraction of a (1,2)-sequence s of length l. Let t be a traversal sequence obtained from a strong traversal sequence t_c of s_c starting at v_c by replacing every* 1 *by* $011(01)^{\lceil l/2 \rceil}$, *and every* 0 *by* $001(01)^{\lceil l/2 \rceil}$. *Then t is a strong traversal sequence for s starting at the left vertex of a pair corresponding to v_c, if v_c is colored* 0, *and at the right vertex of that pair otherwise.*

3.3 Putting Them Together

Propositions 1 and 2 give us a way to construct a universal traversal sequence of length $n^{O(\log n)}$. Given a strong universal traversal sequence t_n for cycles of length n, we construct a strong UTS t_{2n} for cycles of length $2n$ using the construction of Proposition 2 and then the construction of Proposition 1 as follows.

We first replace every 1 in t_n by $011(01)^n$ and every 0 in t_n by $001(01)^n$ as in Proposition 2 to get t_{2n}^{-3}. Then we prepend $(01)^n$ in front of t_{2n}^{-3} to get t_{2n}^{-2}, which is a (1,2)-sequence. Now, we expand every run in t_{2n}^{-2} by $2n = 2n + (2n \bmod 2)$ vertices as in Proposition 1 to get t_{2n}^{-1}. Finally, we prepend 1^{2n} in front of t_{2n}^{-1} to get t_{2n}. We call the prepended sequences $(01)^n$ and 1^{2n} *loaders*.

We claim that t_{2n} is a strong UTS for cycles of length $2n$. Let s be a graph sequence of length $2n$, let $s_\oplus$ be its parity contraction (assuming s is not monochromatic), and let s_c be the pair contraction of $s_\oplus$ (assuming $s_\oplus$ contains at least one pair.) Clearly, $|s_c| \leq n$, so t_n strongly traverses s_c starting at any vertex. The reader may easily verify using Proposition 1, that t_{2n}^{-2} strongly traverses $s_\oplus$ starting at any vertex, and hence using Proposition 2, t_{2n} strongly traverses s starting at any vertex. (Loaders bring us to appropriate vertices so Propositions 1 and 2 are applicable.)

We need a strong UTS for cycles of length (let's say) 5 to start this recursive construction of UTS's.

4 An $O(n^c)$ Universal Traversal Sequence

A disadvantage of the previous construction is the super-polynomial length of the produced UTS. The reason for this length is a pessimism of the construction in two directions. The construction is pessimistic about the length of runs at each step, and about the number of rounds the construction has to be repeated.

A possible way how to reduce the overall length of the resulting UTS is to protect us against appearance of long runs. The way how we achieve this is by introducing two new stages in the construction which break down long runs. One of these stages takes place before parity contraction and the other one before pair contraction. The former stage reduces the length of 0-runs and 1-runs to constant length, the latter one reduces the length of 01-runs to constant length. A consequence of that run length reduction is an expansion of a traversal sequence in Propositions 1 and 2 just by a constant factor.

Both run breaking stages split long runs by inserting extra vertices in them. This actually increases the length of the graph sequence but in the consecutive contraction stages these increases are eliminated. The insertion is done in a way which ensures that any generated traversal sequence behaves on the new graph sequence and on the old one in the same way.

4.1 1-Run and 0-Run Breaking

This stage is aimed at breaking down long 0-runs and 1-runs. We are going to use the following simple proposition for m equal 3 and 5.

Proposition 3. *Let $m \geq 3$ be an odd integer. Then for any $l > m$ there is a k and even d such that $2 \leq d \leq 2m$ and $l = mk + d$.*

Let s be a graph sequence. We replace every run of length five and every run of length longer than six in s by another sequence to obtain a graph s' with all runs of short length. The replacement is done in the following way. Let r be a 0-run or 1-run of length $l = 3k + d \in \{5, 7, 8, 9, \ldots\}$ in s, where k and d are from the previous proposition. If r is a 0-run then we replace it by $r' = (000\ 01)^k 0^d$ otherwise we replace it by $r' = (111\ 10)^k 1^d$. We call this operation *0-run and 1-run breaking* (or simply *1-run breaking.*) We refer to r as to the *original* sequence and to r' as to the *stuffed* sequence.

Lemma 1. *Let s' be obtained from a non-monochromatic graph sequence s by 1-run breaking. Let v be a border vertex of some run in s and let v' be its corresponding vertex in s'. Let (3,6)-sequence t strongly traverse s' starting at v'. Then t, starting at v, completely traverses s, and all vertices in s, except possibly v, are strongly traversed.*

This lemma is proven by focusing attention on behavior of (3,6)-sequences on original and stuffed runs. It can be shown that any (3,6)-sequence t *behaves the same* on any original run r and its corresponding stuffed sequence r' in the following sense. If we run t starting at leftmost (rightmost) *input* vertices v and v' of r and r', respectively, then t keeps us in r as long as it keeps us in r', and it leaves r and r' at the same time on the corresponding *output* vertices, i.e., either at v_l and v'_l, or v_r and v'_r (Fig. 1.) While running t we observe our position only after every whole run in t. (Sequence t may temporarily leave r or r' during a run in t.) More formal treatment is given in the full version of this paper [K1].

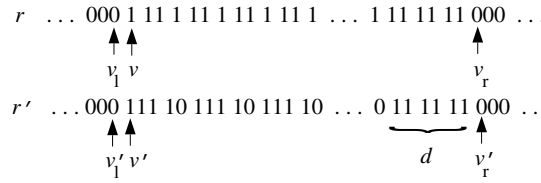


Fig. 1.

Let s and s' be as in Lemma 1. Let t be a traversal (3,6)-sequence which strongly traverses s' starting at any vertex. Let sequence t' be obtained from t by prepending a run w (a loader) of length $|s|$ of opposite color than the first run of t. Then Lemma 1 implies that t' strongly traverses s starting at any vertex.

We conclude that stuffing long runs of 0's and 1's is transparent for (3,6)-traversal sequences. The analysis is done explicitly for (3,6)-sequences but essentially identical analysis can be carried out for any $(2i + 1, 4i + 2)$-sequence, for $i \geq 1$, where we stuff long runs every $(2i + 1)$ vertices.

The following proposition describes behavior of (3,6)-sequences on short runs.

Proposition 4. *1. Any (3,6)-sequence started at left border vertices or right border vertices behaves the same on runs* 11, 1111 *and* 111111*, where the output vertices are vertices of the other color adjacent to these runs in graph sequences that contain these runs. The same holds for* 00, 0000 *and* 000000*.*

2. Any (3,6)-sequence started at left border vertices or right border vertices behaves the same on runs 1 *and* 111*, where output vertices are as above. The same holds for* 0 *and* 000*.*

Let s' be a (1,2)-sequence and t' be a strong traversal sequence of s' starting at a vertex v'. We say that t' *extra strongly traverses* s' if every pair in s' is traversed by t' in such a way that it is entered from outside at one border vertex, walked back and forth on it and then left through the other border vertex.

Lemma 2. *Let s be a non-monochromatic graph sequence with every run of length from $\{1, 2, 3, 4, 6\}$. Let $s_\oplus$ be a parity contraction of s. Let v be a border vertex in s, and $v_\oplus$ its corresponding vertex in $s_\oplus$. Let $t_\oplus$ be an extra strong traversal (1,2)-sequence of $s_\oplus$ starting at $v_\oplus$. Then t, obtained from $t_\oplus$ by replacing every run of length 1 by run of length 3, and every run of length 2 by run of length 6, completely traverses s starting at v.*

Note, if s' is obtained from a non-monochromatic graph sequence s by 1-run breaking, and $s_\oplus$ is obtained from s' by parity contraction, then the length of $s_\oplus$ is at most the length of s.

Let $t_\oplus$ be a (1,2)-sequence which extra strongly traverses any graph (1,2)-sequence of length $|s|$ starting at any vertex. W.l.o.g. the first vertex in $t_\oplus$ has color 0. Let t' be obtained from $t_\oplus$ by the construction from Lemma 2, and let t be obtained from t' by prepending a loader $1^{|s|}$. Then t strongly traverses s starting at any vertex. Moreover, t is a strong UTS for cycles of length $|s|$.

4.2 01-Run Breaking

This stage is aimed at breaking down long 01-runs. The method used here for breaking long 01-runs shares the same spirit with the previous method of breaking 0-runs and 1-runs. Let us describe this stage.

A (5,10)-01-*sequence* is a (1,2)-sequence in which every two consecutive pairs are separated by a 01-run and every 01-run is of length five or ten.

Let s be a graph (1,2)-sequence containing at least one pair. We replace every 01-run in s of length $l \in \{7, 9, 11, 12, 13, 14, \ldots\}$ by a stuffed sequence r' according to the following table. Let $l = 5k + d$ as in Proposition 3. Let $r_0 = 01010$, $r_1 = 10101$, $r'_0 = 01010\ 100100$, and $r'_1 = 10101\ 011011$.

k even:

$$(r_0r_1)^{k/2}(01)^{d/2} \ \rightarrow\ (r'_0r'_1)^{k/2}(01)^{d/2}$$
$$(r_1r_0)^{k/2}(10)^{d/2} \ \rightarrow\ (r'_1r'_0)^{k/2}(10)^{d/2}$$

k odd:

$$(r_0r_1)^{(k-1)/2}r_0(10)^{d/2} \ \rightarrow\ (r'_0r'_1)^{(k-1)/2}r'_0(10)^{d/2}$$
$$(r_1r_0)^{(k-1)/2}r_1(01)^{d/2} \ \rightarrow\ (r'_1r'_0)^{(k-1)/2}r'_1(01)^{d/2}$$

We can establish the following lemma which is an equivalent to Lemma 1. The proof of Lemma 3 looks at 01-runs in s and its corresponding stuffed sequences in s', and shows that (5,10)-01-sequences behave the same on them with respect to vertices $v, v_\mathrm{l}, v_\mathrm{r}$ and $v', v'_\mathrm{l}, v'_\mathrm{r}$ (Fig. 2.)

Lemma 3 (≈1). *Let s' be obtained from a graph (1,2)-sequence s containing at least one pair by 01-run breaking. Let t be a traversal (5,10)-01-sequence ending with a 01-run of length ten. Let v be a left vertex of some pair in s if that pair has the same color as the first vertex of t, or let v be a right vertex of that pair otherwise. Let v' be a vertex corresponding to v in s'. If t extra strongly traverses s' starting at v' then t extra strongly traverses s starting at v.*

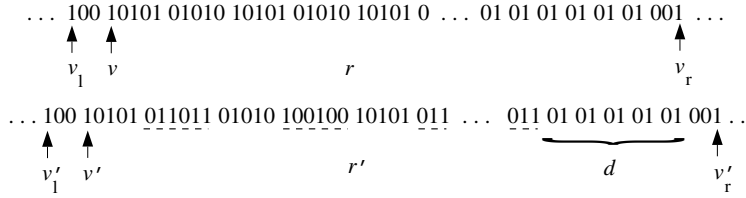


Fig. 2.

Further, similarly to Proposition 4, any (5,10)-sequence beginning with 01-run and ending with a pair started at the leftmost vertices or rightmost vertices behaves the same on sequences 10, 10 10, 10 10 10, 10 10 10 10, and 10 10 10 10 10, where the output vertices are vertices that are neighbors of the surrounding pairs in graph sequences containing these sequences (as in Fig. 2.) The same holds for negated sequences, and for sequences 1, 101 and 10101, and their negations.

We conclude this section with the following lemma.

Lemma 4. *Let s be a (1,2)-sequence containing at least one pair with every 01-run having length in $\{1, 2, 3, 4, 5, 6, 8, 10\}$, and let s_c be the pair contraction of s. Let t_c be a strong traversal sequence of s_c starting at the vertex v_c corresponding to some pair in s. Let v be the left vertex of that pair if the color of that pair is the same as the color of the first vertex in t_c, or let v be the right vertex of that pair otherwise. Let t be obtained from t_c using the following rules:*

vertex	*replace by*
0 followed by 0	*00 10101*
0 followed by 1 (or nothing)	*00 1010101010*
1 followed by 1	*11 01010*
1 followed by 0 (or nothing)	*11 0101010101*

Then t extra strongly traverses s starting at v.

Note, if s' is obtained from a (1,2)-sequence s containing at least one pair by 01-run breaking, and s_c is obtained from s' by a pair contraction, then $|s| \geq 2|s_c|$.

Let t_c be a UTS for cycles of length $\lceil |s|/2 \rceil$. W.l.o.g. the first vertex in t_c has color 1. Let t' be obtained from t_c by the construction from Lemma 4, and let

t be obtained from t' by prepending a loader $0(10)^{\lfloor |s|/2 \rfloor}$. Then t extra strongly traverses s starting at any vertex. Moreover, t extra strongly traverses any graph (1,2)-sequence of length $|s|$.

We choose here to stuff 01-runs every 5 singletons. Similarly to the 1-run breaking stage, we could choose to stuff 01-runs every $2i+1$ singletons, for $i \geq 1$, and use for traversal a $(2i+1, 4i+2)$-01-sequence.

5 The Construction

Lemmas 4, 3, 2, and 1 combine to give a construction that converts a strong UTS for cycles of length m into a strong UTS for cycles of length $2m$. This is presented explicitly in step 3 of the algorithm below. Here is the algorithm constructing a UTS for cycles of length n. Let $k = \lceil \log_2 n \rceil$.

The Algorithm:

1. Let t_0 be a strong UTS for cycles of length 6, that starts with color 1.
2. Apply the construction of Proposition 2 to t_0 to obtain a sequence t'_1, i.e., replace every 1 with $011(01)^6$ and every 0 with $001(01)^6$, and attach in front of t'_1 sequence $0(10)^6$ to get t_1.
3. For $i = 1, \ldots, k-1$, construct a sequence t'_{i+1} from t_i by applying the following rules:

	replace by
a 00-pair	$[00\,10101]^5\,00(10)^5$
singleton 0	$[00\,10101]^2\,00(10)^5$
a 11-pair	$[11\,01010]^5\,11(01)^5$
singleton 1	$[11\,01010]^2\,11(01)^5$

   Prepend loader $0(10)^{6\cdot 4^i}[11\,01010]^{6\cdot 4^i}11(01)^5$ in front of t'_{i+1} to get a sequence t_{i+1}. (This step is a composition of the constructions from Lemmas 2 and 4.)
4. Apply the construction of Lemma 2 to t_k to obtain a sequence t', and attach sequence 1^n in front of t' to get a sequence t.

Sequence t produced by this algorithm is a universal traversal sequence for cycles of length n. By analyzing the algorithm it is easy to see that we can associate a derivation tree of logarithmic depth with the produced sequence. Every node in that tree corresponds either to a pair or a singleton, and certain nodes correspond to loaders. Hence, all the nodes, except the nodes corresponding to loaders, are of bounded degree. It is straightforward to traverse that tree in depth-first manner and to produce sequence t using only logarithmic space. There is also a uniform NC^1 circuit that, given 1^n, constructs that sequence.

The length of the produced sequence is proportional to the number of nodes at the last level of the derivation tree. The average number of descendants of a node corresponding to a pair or a singleton is at most 25.635. Thus, the number of nodes at i-th level of the tree satisfies the recurrence: $f(i+1) \leq 25.635 f(i) +$

$48 \cdot 4^i + 12$. We set $k = \lceil \log_2 n \rceil$ in the algorithm but it is actually enough to set $k = \lceil \log_{\sqrt{5}} n \rceil$, because it can be shown that any graph sequence shrinks to the one-fifth of its original length during repeating the sequence of operations (1-run breaking, parity contraction, 01-run breaking, pair contraction) twice. Thus the length of the produced sequence is $O(n^{\log_{\sqrt{5}} 25.635}) = O(n^{4.031})$.

There may still be some room for slight improvement. If long 01-runs would be stuffed every seven singletons instead of five during the 01-run breaking and (7,14)-01-sequences would be used for traversal, the length of the produced sequence would possibly be $O(n^{3.989})$.

Acknowledgments

I would like to thank to Eric Allender, Mike Saks, Navin Goyal and Dieter van Melkebeek for helpful discussions and for comments on preliminary versions of this paper. I am grateful to Eric Allender, again, for substantial improvement of the language this paper is written in.

References

[A] Aleliunas, R.: A simple graph traversing problem. Master's Thesis, University of Toronto, (1978) (Technical Report 20)

[AKL+] Aleliunas, R., Karp, R. M., Lipton, R. J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. Proceedings of 20-th FOCS (1979) 218–223

[AAR] Alon, N., Azar, Y., Ravid, Y.: Universal sequences for complete graphs. Discrete Appl. Math **27** (1990) 25–28

[ATWZ] Armoni, R., Ta-Shma, A., Wigderson, A., Zhou, S.: $SL \subseteq L^{4/3}$. Proceedings of 29-th STOC (1997) 230–239

[BBK+] Bar-Noy, A., Borodin, A., Karchmer, M., Linial, N., Werman, M.: Bounds on universal sequences. SIAM J. Comput. **18**(2) (1989) 268–277

[B] Bridgland, M. F.: Universal traversal sequences for paths and cycles. J. of Alg., **8**(3) (1987) 395–404

[BT] Buss, J., Tompa, M.: Lower bounds on universal traversal sequences based on chains of length five. Inf. and Comp. **120**(2) (1995) 326–329

[HW] Hoory, S., Wigderson, A.: Universal traversal sequences for expander graphs. Inf. Proc. Letters **46**(2) (1993) 67–69

[I] Istrail, S.: Polynomial universal traversing sequences for cycles are constructible (extended abstract). Proceedings of 20-th STOC (1988) 491–503

[KPS] Karloff, H. J., Paturi, R., Simon, J.: Universal traversal sequences of length $n^{O(logn)}$ for cliques. Inf. Proc. Letters, **28**(5) (1988) 241–243

[K1] Koucký, M.: Log-space constructible universal traversal sequences for cycles of length $O(n^{4.03})$. ECCC-TR01-13 (2001) `http://www.eccc.uni-trier.de/eccc/`

[K2] Koucký, M.: Universal traversal sequences with backtracking. Proceedings of IEEE Conference on Computational Complexity (2001) (to appear)

[N] Nisan, N.: Pseudorandom Generators for Space-bounded Computation. Combinatorica **12**(4) (1992) 449–461

[NSW] Nisan, N., Szemerédi, E., Wigderson, A.: Undirected connectivity in $O(\log^{1.5} n)$ space. Proceedings of 33-rd FOCS, (1992) 24–29

On Universally Polynomial Context-Free Languages*

Nicholas Tran

Department of Mathematics & Computer Science
Santa Clara University
Santa Clara, CA 95054-0290, USA
ntran@math.scu.edu

Abstract. A language is *universally polynomial* if its intersection with every NP-complete language is in P. Such a language would provide an automatic method for generating easy instances of intractable problems. In this note, we give a complete characterization of universally polynomial languages that are context-free, answering an open question in [4].

1 Introduction

It has been observed that most NP-complete languages have an infinite subset in P, i.e. most of these difficult languages have a nontrivial easy subset [2]. Whether this property holds in general is an open question with important consequences: a negative answer would refute the Isomorphism Conjecture for NP-complete sets, whereas a positive answer would imply that public-key cryptosystems based on NP-complete languages are inherently insecure.

Interestingly, it has also been observed that easy subsets for many NP-complete languages can be obtained by applying a *common* restriction to the problem domain. For example, 3-COLORING, CLIQUE, INDEPENDENT SETS, and PARTITION INTO CLIQUES are NP-complete problems that become polynomially solvable when the domain is restricted to chordal graphs [5]. This observation has led Demaine, Lopez-Ortiz, and Munro to ask whether there exists a *universal* restriction that would always yield an easy subset when applied to any NP-complete problem [4]; the existence of such sets would provide an automatic method for generating easy instances of intractable problems. Specifically, they introduced the concept of *universally polynomial* languages as those whose intersection with any NP-complete language is in P, and the weaker concept of *universally simplifying* languages as those whose intersection with any NP-complete language is not NP-complete. These two types of languages together are called *universally easy* languages. It turns out that C, the set of chordal graphs, is not a universally simplifying language, since the graph-theoretic problem of λ-COLORING remains NP-complete for chordal graphs [3]. Assuming that P $\neq$ NP, it follows that C is not universally polynomial.

* This research is partially supported by an SCU-IBM Faculty Research Grant.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 21–27, 2001.
© Springer-Verlag Berlin Heidelberg 2001

It is easy to see that all finite languages are universally polynomial. In practice, we are interested in nontrivial (i.e. infinite) universally polynomial languages that are efficiently computable. Note also that these concepts are interesting only if P $\neq$ NP; otherwise every languages in NP would be universally polynomial and no language in NP would be universally simplifying. At present, a complete characterization of universally easy languages in P is not known. [4] took a first step towards this goal by characterizing universally easy sets that belong to a natural subclass of P, namely the class of regular languages. Specifically, they showed that

1. all finite (regular) languages are universally polynomial;
2. all sparse (regular) languages are either universally simplifying or polynomial;
3. no nonsparse regular language is universally polynomial if P $\neq$ NP.

In this paper, we give a complete characterization of universally polynomial languages that are *context-free*, answering a problem left open in [4]. We show that the set of universally polynomial context-free languages consists of

1. all context-free languages if P = NP;
2. all sparse infinite context-free languages if P $\neq$ NP but DEXT = NEXT;
3. all finite languages if DEXT $\neq$ NEXT.

We also obtain results for non-context-free languages. Specifically, we show that

1. no co-sparse language in P is universally simplifying;
2. no co-sparse language in NP is universally polynomial if P $\neq$ NP.

The rest of this paper is organized into four sections. Relevant definitions and previously known results are stated in Section 2. The complete characterization of universally polynomial context-free languages appears Section 3 and the results on co-sparse universally easy languages appear in Section 4. Finally, some open questions are given in Section 5.

2 Preliminaries

All languages in this paper are over the binary alphabet $\Sigma = \{0, 1\}$. We use the terms *language* and *set* interchangeably. We assume that the reader is familiar with language classes in the Chomsky hierarchy (e.g. regular and context-free languages) and complexity classes (e.g. P, NP, DEXT = $\bigcup_c$ DTIME(2^{cn}), and NEXT = $\bigcup_c$ NTIME(2^{cn})). It is known that REGULAR and CONTEXT-FREE are subclasses of P. See [6] for more details about these classes.

A is a *universally polynomial* language if for every NP-complete language C, $A \cap C \in P$. Note that every finite languages is universally polynomial, and every language in NP is universally polynomial if P = NP.

B is a *universally simplifying* language if for every NP-complete language C, $B \cap C$ is not NP-complete. Note that no language in NP is universally simplifying

if P = NP. A language is *universally easy* if it is either universally simplifying or universally polynomial.

L is a *sparse* language if the number of strings in L of length at most n is bounded above by a polynomial $p(n)$. L is a *tally* language if $L \subseteq 0^*$. Note that all finite and tally languages are sparse. The existence of tally languages in NP − P has been characterized by the following theorem (due to Hartmanis) [1]:

Theorem 1. *DEXT ≠ NEXT iff there is a tally set in NP − P iff there is a sparse set in NP − P.*

L is a *bounded* language if there are some binary strings w_1, w_2, ..., w_n such that $L \subseteq w_1^* w_2^* \dots w_n^*$. The following characterization of sparse context-free languages is well-known:

Theorem 2 ([8,7]). *A context-free language is sparse iff it is bounded.*

3 Characterization of Universally Polynomial Context-Free Languages

We begin by showing that $\Sigma^* \leq_1^p N$ for every nonsparse context-free language N, and hence none of them can be universally simplifying.

Theorem 3. *No nonsparse context-free language is universally simplifying.*

Proof. Let L be a nonsparse context-free language over the binary alphabet $\Sigma = \{0, 1\}$, and G_L be a Chomsky normal form context-free grammar for L without useless or nonreachable variables. Since L is nonsparse, it is infinite. By the Pumping Lemma for context-free languages, there exists some variable T that can make a derivation of the form:

$$T \Rightarrow^* X_0 T Y_0,$$

where X_0, Y_0 are binary strings, and $|X_0| > 0$ or $|Y_0| > 0$. Suppose in addition that the same variable T can make another derivation of the form:

$$T \Rightarrow^* X_1 T Y_1,$$

where X_1 and Y_1 are binary strings, and $(X_0)^{|X_1|} \neq (X_1)^{|X_0|}$, or $(Y_0)^{|Y_1|} \neq (Y_1)^{|Y_0|}$.

Because G_L contains no useless or nonreachable variables, by iterating the above two derivations, we can generate strings in L of the form

$$u x_1 x_2 ... x_k v y_k ... y_2 y_1 w,$$

for every $k \geq 1$, where u, v, and w are some binary strings, and each pair (x_i, y_i) is either $\big((X_0)^{|X_1|}, (Y_0)^{|Y_1|}\big)$, or $\big((X_1)^{|X_0|}, (Y_1)^{|Y_0|}\big)$.

Define a polynomial-time, 1-1, invertible function f from $\{0, 1\}^*$ to L as follows: given a binary string $z = b_1 b_2 \dots b_n$ of length n,

$$f(z = b_1 b_2 ... b_n) = u x_{b_1} x_{b_2} \dots x_{b_n} v y_{b_n} \dots y_{b_2} y_{b_1} w.$$

Since f is 1-1 and polynomially invertible, the image $f(C)$ of any NP-complete set C under f is also NP-complete. This shows that L has an NP-complete subset and therefore is not universally simplifying.

So it remains to show that for nonsparse context-free languages L, the additional derivation $T \Rightarrow^* X_1TY_1$ exists, where $(X_0)^{|X_1|} \neq (X_1)^{|X_0|}$. We will show the contrapositive: if no such additional derivation exists, then L must be a bounded language and therefore must be sparse [8].

For each variable T of G_L, consider all derivations of the form $T \Rightarrow^* X_kTY_k$, where X_k and Y_k are binary strings (if they exist). If for any pair (i, j), $(X_i)^{|X_j|} = (X_j)^{|X_i|}$ and $(Y_i)^{|Y_j|} = (Y_j)^{|Y_i|}$, then X_i and X_j can be written as $X_i = (X_T)^{e_i}$ and $X_j = (X_T)^{e_j}$, where $|X_T| = gcd(|X_i|, |X_j|)$. Similarly, Y_i and Y_j can be written as powers of some Y_T of length $gcd(|Y_i|, |Y_j|)$. Thus any derivation of T that begins with $T \Rightarrow X_kTY_k$ will eventually generate a binary string of the form $(X_T)^*Z_T(Y_T)^*$. Here Z_T is a binary string, whose derivation tree contains no occurrences of T except at the root.

Now consider a derivation tree for any word in L. For any long enough branch so that it has two occurrences of some variable T, collapse the segment starting with the first and ending with the last occurrence of T and replace it with a node labeled by $(X_T)^*T(Y_T)^*$. Repeat this pruning process until no branch of the tree contains two or more occurrences of some variable. Each leaf of the pruned tree is associated with a nested regular expression of the form $(X_{T_1})^*(X_{T_2})^* \dots (X_{T_m})^*Z(Y_{T_m})^* \dots (Y_{T_2})^*(Y_{T_1})^*$, (which appear on the path from the root), and the variables T_i are all different. The number of such pruned trees as well as the number of labels that can be assigned to each leaf are constants depending only on L. Hence every word in L has the form $w_1^*w_2^* \dots w_k^*$, where k is some constant depending only on L. In other words, L is a bounded context-free language and hence it is sparse. □

In view of Theorem 3, it is unlikely that any nonsparse context-free languages can be universally polynomial. In fact, either all or none of them are.

Theorem 4. *The following are equivalent:*

1. *$P = NP$;*
2. *every nonsparse context-free language is universally polynomial;*
3. *some nonsparse context-free language is universally polynomial.*

Proof. (1) ⇒ (2): if P = NP, then the intersection of any context-free language with an NP-complete language is a language in NP and hence must be in P;

(2) ⇒ (3): obvious;

(3) ⇒ (1): suppose L is a nonsparse context-free language that is universally polynomial. The proof of Theorem 3 shows that L has a subset C which is NP-complete. Hence $L \cap C = C$ is both NP-complete and in P, so P = NP. □

We now turn our attention to sparse context-free languages. Using the same technique, we show that $0^* \leq_1^p L$ for every infinite sparse context-free language L, and hence none of them are likely to be universally polynomial. In fact, either none or all of them are. Theorem 1 is crucial in the proof of this characterization.

Theorem 5. *The following are equivalent:*

1. *DEXT = NEXT;*
2. *every infinite sparse context-free language is universally polynomial;*
3. *some infinite sparse context-free language is universally polynomial.*

Proof. (1) ⇒ (2): iff DEXT = NEXT, then the intersection of any infinite sparse context-free language with an NP-complete set is a sparse set in NP and therefore must be in P;

(2) ⇒ (3): obvious;

(3) ⇒ (1): let L be an infinite sparse context-free language. It follows from the Pumping Lemma for context-free languages that there exist strings x, y, u, v, w such that $|y| + |v| > 0$ and $L' = \{xy^iuv^iw : i \geq 0\}$ is an infinite subset of L. The map $f(0^i) = xy^iuv^iw$ is a polynomial-time isomorphism between 0^* and L' such that for any tally language T, T is in P iff $f(T)$ is also in P.

Given a tally language T in NP, we construct an NP-complete language C_T such that $C_T \cap L = f(T)$ as follows. Define $S = \{wx : |w| = |x| \wedge x \in SAT \wedge wx \notin L\}$. S is clearly in NP. Furthermore, SAT $\leq_m^p$ S via the following reduction: on input x, enumerate strings w of length $|x|$ until $wx \notin L$ is satisfied, and output wx. Since L is sparse, such a string can be found in polynomial time. Furthermore, $x \in$ SAT iff $wx \in S$. Thus S is NP-complete. Note also that $S \cap L = \emptyset$. Now define $C_T = S \cup f(T)$. Clearly C_T is also NP-complete and $C_T \cap f(T) = f(T)$.

Since L is universally polynomial, $f(T)$ is in P for all tally set T in NP. Hence all tally sets in NP are in P, so DEXT = NEXT. □

Combining Theorems 4 and 5 we can now describe precisely the class of universally polynomial context-free languages provided we know the relationships between P vs. NP and DEXT vs. NEXT. Note that DEXT ≠ NEXT implies that P ≠ NP.

Corollary 1. *The class of universally polynomial context-free languages consists of*

1. *all context-free languages if P = NP;*
2. *sparse context-free languages if P ≠ NP but DEXT = NEXT;*
3. *finite languages if DEXT ≠ NEXT.*

4 Universally Easy Co-sparse Languages

We extend the results obtained in Section 3 for non-context-free languages whose complements are sparse. Again, it is unlikely that any of the co-sparse languages in NP is universally polynomial, but if one is, then all of them are.

Theorem 6. *No co-sparse language in P is universally simplifying.*

Proof. Let $L \in P$ be a language whose *complement* is sparse. Define a polynomial-time 1-1 function $f : \Sigma^* \mapsto L$ as follows: on input w, f searches for the lexicographically first string v of length $|w|$, such that $vw \in L$ and outputs vw. Such a string exists and can be found in polynomial time since L is in P and $\overline{L}$ is sparse. Clearly f is one-one.

For any NP-complete language C, its image $f(C) \subseteq L$ is also in NP, since f is polynomially honest, and therefore NP-complete. Hence L is not universally simplifying. □

Theorem 7. *If $P \neq NP$, then no co-sparse language is universally polynomial.*

Proof. Suppose P $\neq$ NP. Let C be an NP-complete language and define $C' = \{xw : |x| = |w| \wedge w \in C\}$. C' is also NP-complete. If there is a co-sparse language X such that $X \cap C' \in$ P, then we show that $C \in$ NP, a contradiction.

Let $|\overline{X}^{\leq n}|$ be bounded by some polynomial n^k. The following is a polynomial-time algorithm for C: on input w, enumerate the first $|2w|^k + 1 < 2^{|w|}$ strings of length $|w|$, and for each such string x, check whether $xw \in X \cap C'$. It is clear that $w \in C$ iff one such test succeeds. □

Corollary 2. *The following are equivalent:*

1. *$P = NP$;*
2. *every co-sparse language in NP is universally polynomial;*
3. *some co-sparse language in NP is universally polynomial.*

5 Conclusions

We have characterized the class of universally polynomial context-free languages. Under the assumption that i) P $\neq$ NP but ii) DEXT = NEXT, an infinite universally polynomial language in P exists. It is not known whether the second assumption is necessary for the existence of such sets. It would also be interesting to investigate the relationship between universally easy languages and NP-complete languages without an infinite P-subset.

References

1. Balcázar, J. L., Díaz, J., and Gabarró, J. *Structural Complexity I*, vol. 11 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.
2. Berman, L., and Hartmanis, J. On isomorphisms and density of NP and other complete sets. *SIAM J. Comput. 6* (1977), 305–322.
3. Bodlaender, H. L., Kloks, T., Tan, R. B., and van Leeuwen, J. λ-coloring of graphs. In *Proceedings of the* 17^{th} *Annual Symposium on Theoretical Aspects of Computer Science* (2000), pp. 395–406.
4. Demaine, E. D., Lopez-Ortiz, A., and Munro, J. I. On universally easy classes for NP-complete problems. In *Proceedings of the* 12^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms* (2001), pp. 910–911.

5. Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, 1979.
6. Hopcroft, J., and Ullman, J. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.
7. Ibarra, O., and Ravikumar, B. On sparseness, ambiguity and other decision problems for acceptors and transducers. In *Proceedings of the 3rd Annual Symposium on Theoretical Aspects of Computer Science* (1986), pp. 171–179.
8. Latteux, M., and Thierrrin, G. On bounded context-free languages. *Elektron. Informtionsverarb. Kybernet. 20* (1984), 3–8.

Separating Oblivious and Non-oblivious BPs

Kazuo Iwama[1], Yasuo Okabe[1], and Toshiro Takase[2]

[1] School of Informatics, Kyoto University
Yoshida-honmachi, Sakyo-ku Kyoto 606-8501, Japan
{iwama, okabe}@kuis.kyoto-u.ac.jp
[2] IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma Yamato 242-8502, Japan
E30809@jp.ibm.com

Abstract. This paper gives an exponential separation on the depth of branching programs (BPs) between oblivious and non-oblivious BPs. Namely, there is a difference just like the difference between sequential and NC computation: (i) There is a Boolean function f_1 of N variables which can be computed by a polynomial-size, syntactic BP with a depth of $2\log N - \log\log N + 1$ but cannot be computed by any oblivious BPs with a depth of $(2-\varepsilon)N$ for some $\varepsilon \in o(1)$. (ii) Similarly, there is an f_2 computed by a syntactic depth of $\log^3 N$ but not by an oblivious depth of $\Omega(N \log N)$. (iii) We also show that any (unrestricted) BP of depth t can be simulated by an oblivious BP with a depth of $N + \lceil (t - \log N)/(\log\log N + C) \rceil \cdot N$. The third result implies that f_1 cannot be computed by any BP with a depth less than $\log N + \log\log N$ and f_2 not with a depth of $o(\log N \cdot \log\log N)$. Note that most bounds in this paper include constant factors and lower-degree terms.

1 Introduction

One of the primary goals of theoretical computer science is to prove non-linear lower bounds for the size of Boolean circuits. However, in spite of serious efforts for several decades, we are still at a very elementary stage and are struggling for good research hints. Probably for this reason, several researches have been shifting their attention from Boolean circuits to Boolean *branching programs* (BPs): (i) BPs are also a general computation-model whose size and depth are naturally related to space and time [8]. (ii) It is relatively easy to consider reasonable restrictions which provide interesting subclasses, such as oblivious models discussed in this paper, for more attainable lower-bound arguments.

Not surprisingly, therefore, we have a large literature. First of all, Ajtai [2] recently made a big breakthrough by showing a nonlinear depth lower bound for unrestricted BPs (no nontrivial lower bounds for even read-twice BPs were known before.). In [13], Thathachar gave an exponential separation on the size between k-sBPs (read-k syntactic BPs) and $(k+1)$-sBPs for all k, extending the previous result [7] that separates only between $k = 1$ and $k = 2$. Then, Beame, Saks and Thathachar [10] obtained the first (again exponential on the size) separation result between semantic and syntactic BPs, i.e., between k-BPs

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 28–38, 2001.
© Springer-Verlag Berlin Heidelberg 2001

and k-sBPs, by giving a Boolean function f_k which can be computed by a 2-BP of polynomial size but cannot be computed by any k-sBP of polynomial size. For oBPs (oblivious BPs) and OBDDs, similar separation results have been known for longer time, including those between k-oBPs and $(k+1)$-oBPs [13], between k-OBDDs and $(k+1)$-OBDDs [9,3,1] and between 2-oBPs and $\bigcup_k$ k-OBDDs [11]. (The separation between k-oBPs and $(k+1)$-oBPs is mentioned only implicitly in [13].)

In this paper, we focus our attention not on the size but on the depth of BPs which is an equivalently popular measure (e.g., [4,6]). Interestingly, our results have exactly the same flavor as typical results on parallel vs. sequential computation. Remember what we have usually done in the research of parallel computation: (i) For a given problem (or a Boolean function) Q, we design an efficient (poly-log depth) parallel algorithm A and then (ii) claim its efficiency by comparing the time complexity of the standard sequential simulation of A and the time complexity of the best known sequential algorithm for Q. This paper just follows this approach: (i) There is a Boolean function f_1 of N variables which can be computed by a syntactic BP of depth $2\log N - \log\log N + 1$ but cannot be computed by any oblivious BPs of depth $(2-\varepsilon)N$ for some $\varepsilon \in o(1)$. Also, (ii) there is an f_2 which is computed by a syntactic depth of $\log^3 N$ but not by an oblivious depth of $\Omega(N \log N)$. Thus, a poly-logarithmic depth is enough for some functions which need a polynomial (i.e., exponentially larger) oblivious depth. Our third result is on the simulation; (iii) any BP of depth t can be simulated by an oblivious BP with a depth of $N + \lceil (t - \log N)/(\log\log N + C) \rceil \cdot N$. This simulation result implies that f_1 cannot be computed by any BP with a depth of less than $\log N + \log\log N$ and f_2 not with a depth of $o(\log N \cdot \log\log N)$. Thus those lower bounds are somewhat close to the upper bounds in (i) and (ii) above, but are not tight. It should be noted that our results in this paper give specific constant factors and even lower-degree terms.

As for lower-bounds on the depth of oblivious BPs, three Boolean functions have appeared in the literature: two have an $\Omega(N \log N)$ lower bound [4] and the third one has an $\Omega(N \log^2 N)$ lower bound [6]. Our second function f_2 is a modification of one of the two functions in [4]. Similar modification is also possible to the function in [6] and we can design an efficient syntactic BP. Unfortunately, we can only get an $O(N^{\varepsilon})$ upper bound for this syntactic depth at this moment (see Section 5).

It is hard to cover all the related research in this detailed abstract. Among others, many authors have discussed BPs which can read most variables only once but can read the remaining few variables many times [e.g., [12]]. OBDDs have a long history of research which includes a lot of papers discussing their practical merits for computer-aided design and hardware verification [e.g., [5]].

In the rest of this paper, we first give definitions and notations in the next section. Then we give the simulation result in Section 3, which will help understanding basic differences between oblivious and non-oblivious BPs. Sections 4 is for the two separation results and finally we briefly mention some related results in Section 5.

2 Definitions

Fig. 1 illustrates a simple example of a BP. A BP is an acyclic directed graph with a single *source vertex* of zero in-degree. Each *sink vertex* of zero out-degree is labeled either 0 or 1. Each non-sink vertex is labeled some variable x_i among $x_1, x_2, \ldots, x_N$ and has two outgoing edges labeled 0 and 1. Once a 0/1-assignment to the variables x_1 through x_N is given, we can determine a unique *computation path* from the source vertex to some sink vertex labeled 0 or 1. For example, if $(x_1, x_2, x_3) = (1, 0, 1)$, the unique path ends with the sink vertex labeled 1 in Fig. 1. Thus a BP computes a Boolean function of N variables. The *size* of a BP is the number of non-sink vertices and the *depth* of a BP is the number of non-sink vertices on the longest computation path. In this paper, all the BPs have a size bounded by some polynomial in N, so we often omit mentioning it.

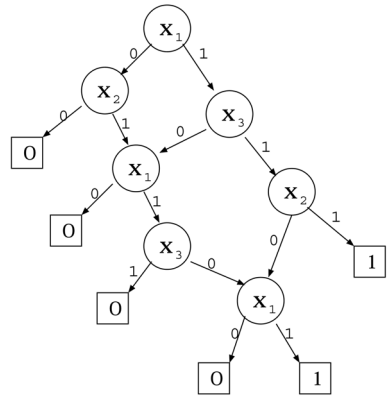


Fig. 1. example of a BP

As shown in the example in Fig. 1, each computation path can read the variables in a different order. If all the computation paths read the variables in the same order, such a BP is called an *oblivious* BP. (*Unrestricted* or *semantic*) BPs and *syntactic* BPs have no such restriction on the order of reading variables. Instead there is an important difference between them in the way of measuring the depth. See Fig. 1 again: If one looks at this BP carefully, it turns out that the path $x_1x_2x_1x_3x_1$ is not actually followed by the BP since it would mean that the value of x_1 is 0 when it is read for the first time and is 1 for the second time. Such a computation path is said to be *not realizable*. Syntactic BPs take into account all computation paths whether or not they are realizable and its depth is the length of the longest path among them. Semantic BPs (or simply BPs) take into account only realizable computation paths and its depth is the length of the longest, realizable path.

A BP is said to be *tree-like* if each vertex (other than the source) has only one in-degree. Any BP can be changed to a tree-like BP simply by splitting a vertex of in-degree k into k vertices of in-degree one from top to bottom. Note that the depth does not change but the size can enlarge greatly. Obviously, any tree-like BP has an equivalent tree-like BP such that no variable appears twice or more in every path.

3 Simulation by Oblivious BPs

Recall that the relationship between BPs and oblivious BPs is somewhat similar to the relationship between parallel and sequential computation.

Theorem 1. Suppose that a Boolean function f of N variables is computed by a BP of depth t. Then f can be computed by an oblivious BP of depth $N + \lceil (t - \log N)/(\log\log N + C) \rceil \cdot N$.

Proof. For a given BP Q of depth t we construct an oblivious BP P. The construction is basically to divide Q into blocks of depth $\log\log N$ and simulate each of them by oblivious BPs of depth N. However, a special treatment can be done for the top $\log N$-depth portion of Q (see Fig. 2).

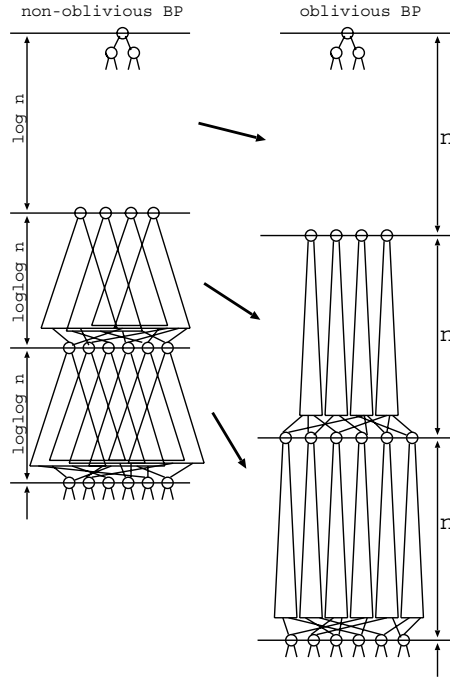


Fig. 2. Simulation by Oblivious BPs

Lemma 1. Any BP of depth $\log N$ can be simulated by an oblivious BP of depth N.

Proof. We first change the given BP, say, M_1, into a tree-like BP, M_2, which includes $N-1$ vertices or at most $N-1$ variables. Suppose that M_2 looks like Fig. 3(a). Then an equivalent oblivious BP can be constructed simply by "inserting missing vertices" in each path of M_2 as shown in Fig. 3(b). For example, we insert v_2, v_4, v_5 and v_7 into the path of $v_1v_3v_6$ of M_2. Each inserted vertex has only one out-degree and therefore the size dose not blow up. □

It should be noted that this construction is not always possible if the variable order in the target oblivious-BP is restricted. For example, if the variable order of the oblivious BP is the opposite of the above width-first order, then the size turns out to be exponential. This is the reason why we cannot divide Q into all $\log N$-depth blocks.

Lemma 2. Let $C > 0$ be any constant and $S = v_1v_2 \cdots v_N$ be any variable order. The any BP of depth $\log\log N + C$ whose variable set is a subset of $\{v_1, v_2, \cdots, v_N\}$ can be simulated by an oblivious BP of depth N whose variable order is S.

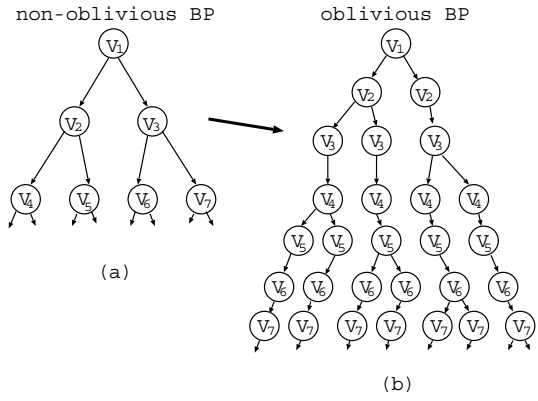


Fig. 3. Basic idea of simulation

Proof. Since the given BP, say, M_1, is of depth $\log\log N + C$, the number of vertices is $O(\log N)$, namely M_1 looks at only $O(\log N)$ variables. Hence we can immediately make an oblivious BP, M_2, whose shape is a completely binary tree of size $2^{O(\log N)}$ and whose variable order dose not contradict to S. Finally we can just insert missing variables as before. □

Now we divide the original BP Q into blocks as shown in Fig. 2. The first level includes vertices from the first level until the $(\log N+1)$th level, the second block includes vertices from the $(\log N+1)$th level until the $(\log N+\log\log N+C+1)$th level. (Namely, vertices at the $(\log N + 1)$th level are shared by the first and the second blocks.) In general the kth ($k \geq 3$) block starts at $(\log N + (k - 1)(\log\log N + C) + 1)$th level and ends at $(\log N + k(\log\log N + C) + 1)$th level.

Thus Q is divided into one block of depth $\log N$ (vertices at the lowest level at each block are viewed as sink nodes and are excluded in counting the depth), and $\lceil (t - \log N)/(\log\log N + C) \rceil$ blocks of depth $\log\log N + C$, excepting the last block which may be shorter.

Each block can be viewed as a set of (a polynomial number of) BPs. Each of them is transformed into a tree-like BP but we do not change vertices at the lowest level, i.e., they can accept an arbitrary number of incoming edges. Now these BPs are transformed into oblivious BPs as follows:

(1) The first block is transformed into an oblivious BP using the construction in Lemma 1.

(2) For each BP in other blocks, we use the construction in Lemma 2, where the variable sequence S for all these BPs must be the same but may be any (fixed) one.

Note that we did not change the vertices which are shared by neighboring two blocks (whose labels may have been changed) and therefore those transformed BPs are naturally combined into a single BP which is equivalent to Q. One can see that this BP is oblivious and its depth is $N + \lceil (t - \log N)/(\log\log N + C) \rceil \cdot N$. □

4 Separation Results

In this section $\log N$ and $\log\log N$ always denote $\lceil \log N \rceil$ and $\lceil \log\log N \rceil$, respectively. We introduce two Boolean functions which give an exponential separation on the depth between oblivious and non-oblivious BPs.

Theorem 2. Let ε be any function such that $\varepsilon > 2\frac{\log\log N}{\log N}$ for infinitely many integers $N \geq 0$ (e.g., ε may be $\frac{(\log\log N)^{1+\delta}}{\log N} \in o(1)$). Then there exists a Boolean function, f_1, of N variables such that f_1 can be computed by a syntactic BP of depth $2\log N - \log\log N + 1$ but cannot be computed by any oblivious BP of polynomial size whose depth is $(2 - \varepsilon)N$.

Proof. A Boolean function $f_1(x_1, x_2, \cdots, x_N)$ is defined as follows: Its variables $x_1, x_2, \cdots, x_N$ are partitioned into blocks $\beta_0, \beta_1, \cdots, \beta_{\lceil \frac{N}{\log N} \rceil - 1}$. The first block β_0 consists of the first $\log N - \log\log N$ variables, i.e., x_1 through $x_{\log N - \log\log N}$, and each of subsequent β_i's ($i \geq 1$) consists of subsequent $\log N$ variables (the last one may be shorter again). To decide the value of f_1: (i) We first look at block β_0 and regard its value as a binary number b_0. (ii) Then we go to block β_{b_0+1} and again regard its value as a binary number b_1. (iii) The value of f_1 is equal to the value (0 or 1) of x_{b_1}.

Upper Bound. It is not hard to see that f_1 can be computed by a tree-like BP Q of depth $2\log N - \log\log N + 1$. Its size is obviously bounded by a polynomial because its depth is logarithmic. Q first reads variables in β_0 which includes $\log N - \log\log N$ variables and due to its value Q decides which β_i of $\log N$ variables should then be read. Finally Q reads x_{b_1} and goes to a sink vertex of 0 or 1.

Lower Bound. Suppose that there is an oblivious BP, M, of depth $(2 - \varepsilon)N$ whose variable sequence is S. For simplicity we assume that the length of S is

even and let S_1 and S_2 be the first and the second halves of S. The following simple lemma is important :

Lemma 3. There are at least $\frac{\varepsilon}{2}N$ variables which appear only in S_1 and also at least other $\frac{\varepsilon}{2}N$ variables which appear only in S_2.

Proof. Since $|S_1| = (1 - \frac{\varepsilon}{2})N$, there are at least $\frac{\varepsilon}{2}N$ variables which do not appear in S_1. Those variables must appear in S_2 since f_1 clearly depends on all the variables. Similarly for S_2. □

Let us call variables only appearing in S_1 *A-variables* and variables only in S_2 *Z-variables*. Recall that $|\beta_0| = \log N - \log\log N$ and there are at least $\frac{\varepsilon}{2}N$ Z-variables. Therefore, some block, $\beta_i (i \geq 1)$, contains at least

$$\frac{\frac{\varepsilon}{2}N - (\log N - \log\log N)}{\lceil \frac{N}{\log N} \rceil - 1} = \frac{\varepsilon}{2}\log N - O(\frac{\log^2 N}{N})$$

Z-variables. Although details are omitted, we can show that the second term $(O(\frac{\log^2 N}{N}))$ is so small and does not cause any problem if we omit it. In the following we do not write it just for simplicity. Select arbitrary $\frac{\varepsilon}{2}\log N$ Z-variables in β_i and let BZ_i be the set of those variables. Also let $\overline{BZ_i}$ be the set of the remaining $(1 - \frac{\varepsilon}{2})\log N$ variables in β_i. Recall that the value assigned to β_i-variables is a pointer to some variable $x_i (1 \leq i \leq N)$. Hence we can define the following equivalence classes on the set of the whole N variables:

$$E_j = \{x_i | \text{ the pointer to } x_i \ (= \text{the value } i) \\ \text{has the same } 0/1 \text{ assignments } (= j) \text{ to } \overline{BZ_i} \text{ variables } \}$$

Note that the number of these equivalence classes is $2^{(1-\frac{\varepsilon}{2})\log N} = N^{1-\frac{\varepsilon}{2}}$, and therefore at least one E_j contains $\frac{\frac{\varepsilon}{2}N}{N^{(1-\frac{\varepsilon}{2})}} = \frac{\varepsilon}{2} \cdot N^{\frac{\varepsilon}{2}}$ A-variables or more.

Now recall that all those A-variables appear in S_1 which is the first half of the variable sequence, and any of them, x_k, can be selected by assigning some proper values to BZ_i variables. (Reason: We first Fix the β_0 variables to show the β_i block. Then fix $\overline{BZ_i}$ variables so that it will show the E_j equivalence class which includes x_k. Now we finally fix BZ_i variables to show x_k.) Since those A-variables appear in S_1 and all the BZ_i variables appear in S_2, the BP M has to remember all the values of the A-variables at the end of S_1. This needs $2^{\frac{\varepsilon}{2} \cdot N^{\frac{\varepsilon}{2}}}$ vertices there, which is not bounded by any polynomial if ε satisfies the condition of the lemma. □

Theorem 3. There exists a Boolean function, f_2, of N variables such that f_2 can be computed by a syntactic, read-once BP of depth $\log^3 N$ but needs an $\Omega(N \log N)$ depth to be computed by oblivious BPs.

Proof. Before giving the definition of f_2, we shall review the lower-bound result in [4].

Lemma 4 [4]. There is a Boolean function, f_2^*, of N variables for which any oblivious BP need a depth of $\Omega(N \log N)$.

Boolean function f_2^* looks like this: The variables are $x_1, \cdots, x_n, y_1, \cdots, y_n$ and each variable takes value 0,1 or 2 i.e., they are 3-way variables. Actually, we use two Boolean variables for each x_i and y_i, and hence $N = 4n$. It is defined that

$f_2^* = 1$ iff the 0/1-sequence of $x_1, \cdots, x_n$ obtained by neglecting the variables having value 2 is equal to the 0/1-sequence of $y_1, \cdots, y_n$. The basic idea of their proof is as follows: Suppose that an oblivious BP, M, computes f_2^* whose variable sequence is S and that most variables are read at least k times in S. Then it turns out, by a Ramsey-like theorem, that we can select $(\frac{1}{2})^{ck}n$ x_i-variables and the same number of y_i-variables for some constant $c > 0$ such that S is divided into ck blocks each of which contains only selected x_i-variables or only selected y_i-variables. Since all the other variables can be killed by assigning 2, a simple counting argument gives us

$$W^{ck-1} \geq 2^{n(\frac{1}{2})^{ck}},$$

where W is the maximum width at each boundary of blocks. For W to be bounded by a polynomial, we need that $k = \Omega(\log n)$, which implies the lower bound of the lemma.

This argument is perfectly good for their result but a careful observation gives us the following improvement:

Observation. We only need less survived variables than $n(\frac{1}{2})^{ck}$ ones to obtain a superpolynomial bound. Namely $\log^2 n$ ones are enough since in order to satisfy

$$W^{ck-1} \geq 2^{\log^2 n},$$

W needs to go beyond a polynomial if $k = o(\log n)$.

This observation seldom improves their result (only a constant factor), but it is very important in this paper. We again use a similar addressing scheme as before to show these surviving variables.

Definition of the Function. $f_2(X_1, X_2)$ is defined as follows: Both X_1 and X_2 consist of $n \log N' + N'$ variables like

$$X_i = Q_{i,1}, ..., Q_{i,n}, x_{i,1}, ..., x_{i,N'} \quad (i = 1, 2),$$

where $n = \log^2 N'$ and each $Q_{i,j}$ includes $\log N'$ variables. Note that the whole number N of variables is $N = 2(n \log N' + N')$. When 0/1-values are assigned to the variables, let $s_{i,j}$ $(i = 1, 2)$ be the sum of the values (the binary numbers as before) for $Q_{i,1}, \cdots, Q_{i,j}$. Then, $f_2 = 1$ if and only if (i) the value for $Q_{i,j} > 0$ for all i and j, (ii) $s_{i,n} \leq N'$ for both $i = 1$ and 2, and (iii) $x_{1,s_{1,j}} = x_{2,s_{2,j}}$ for all $1 \leq j \leq n$. Namely the 0/1-sequence of $x_{1,j}$'s which are selected by $Q_{1,1}, ..., Q_{1,n}$ must be the same as the 0/1-sequence of similarly selected $x_{2,j}$'s.

Lower Bound. We omit the lower-bound proof since it is very similar to Lemma 4.

Upper Bound. Since the previous BP given in the proof of Theorem 2 had a logarithmic depth, it was able to be constructed as a tree-like BP. This time, however, tree-like BPs obviously blow up and we need a careful description for DAG-like BPs. To this end, we introduce what we call a "RAM-model" M for BPs: (i) M has a level-counter i and an internal state S of $O(\log N)$ bits. (ii) Its one-step move is determined by the values of i and S, which includes reading some variable, and changing the values of i and S due to the current values of i

and S and the value of the variable just read. (iii) M may read more than one variable in its one-step move (multiple read).

It is not hard to see that a program written by this rule can be immediately transformed to a BP by changing all different states into different vertices. Its size is bounded by a polynomial since the internal state has only $O(\log N)$ bits. Its depth is the maximum level, but if multiple read is used then we have to increase it appropriately.

Now we give our RAM-model program M which computes f_2. M reads variables in the following order:

$$Q_{1,1}, x_{1,s_{1,1}}, Q_{2,1}, x_{2,s_{2,1}}, Q_{1,2}, x_{1,s_{1,2}}, ..., Q_{1,n}, x_{1,s_{1,n}}, Q_{2,n}, x_{2,s_{2,n}}$$

Here which variable $x_{i,s_{i,j}}$ is depends on the value of $Q_{i,j}$. Also, all variables in $Q_{i,j}$ are read in one step, which will not cause any problem since $|Q_{i,j}| = \log N' \leq \log N$. Internal state of M has the form of $S = (P_1, P_2, z)$, where P_1 holds the position of variable $x_{1,j}$ which was read most recently, i.e., that is the value of $s_{1,j}$ at that moment. z needs only one bit to hold the value of $x_{1,s_{1,j}}$ until it is compared with $x_{2,s_{2,j}}$. Thus, the size of S is $2\log N' + 1$ or $O(\log N)$. The level counter i changes as follows:

$$(1,1)_Q(1,1)_x(2,1)_Q(2,1)_x(1,2)_Q(1,2)_x \cdots (2,n)_Q(2,n)_x,$$

where $(i,j)_Q$ means that it is the time when M is to read $Q_{i,j}$ and $(i,j)_x$ means that it is the time when M is to read $x_{i,s_{i,j}}$. Note that its initial state is $(0,0,0)$.

Now we describe a one-step behavior at each level.

(i) The current level is $(i,j)_Q$. Read $Q_{i,j}$. Suppose that the current state is (P_1, P_2, z). Then if $Q_{i,j} = 0$ or $P_i + Q_{i,j} \geq N'$, then go to [0] (= a sync-node labeled by 0). Otherwise, change its state to $(P_1 + Q_{1,j}, P_2, z)$ when $i = 1$ and to $(P_1, P_2 + Q_{2,j}, z)$ when $i = 2$ and go to the next level.

(ii) The current level is $(1,j)_x$. Read x_{1,P_1}, change its state to (P_1, P_2, x_{1,P_1}), and go to the next level.

(iii) The current level is $(2,j)_x$. Read x_{2,P_2}. If the current z is equal to x_{2,P_2} then if $j = n$ then go to [1] else just go to the next level. If $z \neq x_{2,P_2}$ then go to [0].

One can see that this program reads at most $2n$ $Q_{i,j}$'s and $2n$ $x_{i,j}$'s and hence the depth of the equivalent BP is $2n\log N' + 2n = O(\log^3 N)$. (Recall that $n = \log^2 N'$ and $N = (n\log N' + N')$.) Also this BP never reads a single variable twice, namely, it is a read-once BP. □

5 Some Remarks

Here are a few remarks which might help:

Remark 1. One might have noticed that there is a trivial function of N variables which is computed by a BP of depth $\log N$ but needs an N oblivious depth. One such example is the function which is computed by the BP that has a form of a complete binary tree and all the vertices read different variables.

The final value is the value of the variable which is read at the end of each path. Clearly this function depends on all the variables and therefore any equivalent oblivious BP needs the depth which is equal to the number of variables. Such a separation is optimal due to Theorem 1. Our real purpose in this paper is not to give such a single example (which is enough only to prove a separation) but to give more general evidence that oblivious BPs need exponentially more depth than their non-oblivious counterparts for a wide range of Boolean functions.

Remark 2. By a small modification of f_2 of Theorem 3, we can obtain a function f_2' which needs a non-linear oblivious depth but can be computed by a BP of depth $\log^{2+\varepsilon} N$ for any constant $\varepsilon > 0$. The idea is that if k is a constant, then the width W which satisfies $W^{ck-1} \geq 2^{(\log n)^{1+\varepsilon}}$ is not bounded by any polynomial. Thus the construction of f_2' is exactly the same as f_2 except for $n = \log^{1+\varepsilon} N'$.

Remark 3. The best depth lower bound for oblivious BPs is $\Omega(N \log^2 N)$ in [6]. The Boolean function used there also uses 3-way variables, i.e., variables having value 2 are all "killed." Therefore, we can also build our addressing scheme into this function. Unfortunately, their lower-bound proof needs a polynomially many variables to survive. Thus this construction still needs $O(N^{\varepsilon})$ depth to be computed by unrestricted BPs.

6 Conclusion

An apparently interesting attempt is to try to find a nontrivial separation between oblivious and non-oblivious depth which is *optimal*. As mentioned before, our Theorems 2 and 3 are not optimal with respect to Theorem 1. Theorem 3 is a little far from the optimal value but Theorem 2 is much closer. One possibility is to reduce the size of β_i $(i \geq 1)$ from $\log N$ to, say, $(\log \log N)^k$. If we do so, then it can no longer make pointers to all the variables. However, one can see that it would be enough to have pointers to some subset of the variables which contains sufficiently many A-variables. If the numbers of A- and Z-variables are relatively large, e.g., some $\frac{1}{c}$ fraction of the total number for a constant C, this appears to be quite plausible.

In this paper we were only able to give a similar type (having the addressing scheme) of Boolean functions for the separation. Seeking different types of Boolean functions having the same separation property will be also interesting.

References

1. F. Ablayev. "Randomization and nondeterminism are comparable for ordered read-once branching programs" Proc. 24th ICALP, LNCS 1256, 195-202, 1997.
2. M. Ajtai. "A Non-linear Time Lower Bound for Boolean Branching Programs" Proc.40th IEEE FOCS, pp.60-69, 1999.
3. F. Ablayev and M. Karpinski. "On the power of randomized branching programs " Proc. 23th ICALP, LNCS 1099, 348-356, Paderbon, Germany, 8-12 July 1996.
4. N. Alon, and W. Maas, "Meanders, Ramsey theory and lower bound for branching programs" Proc.27th IEEE FOCS, pp.410-417, 1986.

5. R. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean function with application to integer multiplication." IEEE Trans. Computers, C-40(2), pp. 205-213, 1991.
6. L. Babai, N. Nisan, M. Szegedy, "Maltiparty protocols, pseudorandom generators for logspace, and time-space trade-offs" J. Computer System Sciences 45, pp. 204-232, 1992.
7. A. Borodin, A. Razborov, R. Smolensky, "On lower-bounds for read-k-times branching programs" Computational Complexity, 3, pp. 1-18, 1993.
8. R. Boppana and M. Sipser. "The complexity of finite functions" In handbook of Theoretical Computer Science (Volume A), Ed. Jan van Leeuwen. Elsevier and MIT Press, 1990.
9. B. Bolling, M. Sauerhoff, D. Sieling, and I. Wegener, "Hierarchy theorems for kOBDDs and kIBDDs." Theoretical Computer Science, 1996.
10. P. Beame, M. Saks and J. Thathachar, "Time-Space Tradeoffs for Branching programs," Proc. 39th IEEE FOCS, pp.254-263,1998.
11. M. Krause "Lower Bounds for Depth-Restricted Branching Programs," Information and Computation 91, pp. 1-14, 1991.
12. P. Savicky and S. Zak, "A Hierarchy for $(1, +k)$-Branching Programs with Respect to k," Proc. MFCS97 (LNCS 1295), pp478-487, 1997.
13. J. Thathachar, "On separating the read-k-times branching program hierarchy" Proc. 30th ACM STOC, 1998.

Program Schemes, Queues, the Recursive Spectrum and Zero-One Laws

Iain A. Stewart

Department of Mathematics and Computer Science,
University of Leicester, Leicester LE1 7RH, U.K.
i.a.stewart@mcs.le.ac.uk
http://www.mcs.le.ac.uk/~istewart

Abstract. We prove that a very basic class of program schemes augmented with access to a queue and an additional numeric universe within which counting is permitted accepts exactly the class of recursively solvable problems. The class of problems accepted when access to the numeric universe is removed is exactly the class of recursively solvable problems that are closed under extensions. We build upon NSPQ(1) an infinite hierarchy of classes of program schemes and show that the class of problems accepted by these program schemes has a zero-one law and consists of those problems defined in any vectorized Lindström logic formed using operators whose corresponding problems are recursively solvable and closed under extensions. We apply our results to yield logical characterizations of complexity classes and provide logical analogs to inequalities and hypotheses from complexity theory involving the classes **NP** through to **ELEMENTARY**.

1 Introduction

Descriptive complexity theory essentially began with Fagin's Theorem [7] which states that a problem can be accepted by a polynomial-time non-deterministic Turing machine iff it can be defined in existential second-order logic. Since then, the relative relationships of a whole range of complexity classes and logics have been examined (see, *e.g.*, [6,10]). Having logical equivalents of hypotheses in complexity theory enables attack using tools from finite model theory, such as Ehrenfeucht-Fraïssé games and zero-one laws.

The connection between a complexity class and a logic need not be as tight as in Fagin's Theorem. For example, it is known that a problem is in **P** (resp. **PSPACE**) iff it can be defined in inductive (resp. partial) fixed-point logic, but in both cases only when we work with ordered finite structures (see [6,10]). It turns out that on the class of all finite structures there are (computationally trivial) problems not definable in these fixed-point logics. Having logical characterizations of complexity classes on the class of all finite structures is preferable. For instance: ordered structures are more difficult to work with when applying the inexpressibility tools of finite model theory; in applications of descriptive complexity theory, *e.g.*, to databases, it is almost always the case that finite structures are not ordered; and it is often undecidable as to whether a formula is well-formed when restricted just to ordered finite structures.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 39–48, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Whilst it remains unknown as to whether there is a logic capturing **P** on the class of all finite structures, Abiteboul and Vianu proved the following result [2,3]: **P** = **PSPACE** iff inductive fixed-point logic and partial fixed-point logic are equally expressive on the class of all finite structures. It is particularly interesting that a complexity-theoretic statement like **P** = **PSPACE** has an equivalent formulation in logic where there is a such a mismatch between the complexity class and the class of problems defined in the respective logic. Abiteboul, Vardi and Vianu [1] and Dawar [5] subsequently obtained similar results involving a range of complexity classes including **P**, **NP**, the Polynomial Hierarchy **PH**, **PSPACE** and **EXPTIME**. One thing all of their logics have in common is that they can be realised as fragments of bounded-variable infinitary logic $\mathcal{L}^{\omega}_{\infty\omega}$.

In obtaining their result, Abiteboul and Vianu developed a model of computation known as a loosely coupled generic machine (renamed a relational machine in [1]). Abiteboul and Vianu are not the only researchers in recent times to work with models of computation taking finite structures as inputs. Perhaps the best known such computational model is the abstract state machine of Gurevich [9]. Recently, the expressive power of different classes of program schemes has been considered. Program schemes are similar in flavour to relational machines and abstract state machines, and form a model of computation that is closer to the notion of a high-level program than a logical formula is; yet program schemes remain amenable to logical manipulation. In [4,11,12,13], the computational power of different classes of program schemes (on the class of all finite structures) is compared with the expressive power of certain logics. A crucial difference between program schemes and the traditional logics of descriptive complexity theory is that program schemes can be augmented with 'computational' constructs not really available in logic, such as stacks and arrays.

In this paper, we augment a very basic class of program schemes NPS(1) with a queue and an extra 'numeric' universe. We prove that when both augmentations are present, the class of problems accepted by the resulting class of program schemes NPSQ$_+$(1) is exactly the class of recursively solvable problems, *i.e.*, the class of problems $\mathcal{REC}$; and when only the queue is added, the class of problems accepted by the resulting class of program schemes NPSQ(1) is exactly the class of recursively solvable problems that are closed under extensions, *i.e.*, the class of problems $\mathcal{REC} \cap \mathcal{EXT}$. We proceed to introduce universal quantification into our program schemes by constructing an infinite hierarchy NPSQ of classes of program schemes obtained by interleaving the application of universal quantification with the constructs of the program schemes of NPSQ(1). It turns out that the class of problems accepted by the program schemes of NPSQ has an alternative 'semantic' characterization as the class of problems defined by the sentences of any vectorized Lindström logic for which the problem corresponding to the operator is recursively solvable and closed under extensions. It follows as a corollary that this class of problems has a zero-one law. We remark that by [11], there are problems in NPSQ(1)$\cap$**NP** that are not definable in $\mathcal{L}^{\omega}_{\infty\omega}$.

The fact that NPSQ$_+$(1) is computationally complete enables us to restrict the resources (time and space) used by these program schemes so that we can

logically capture many complexity classes ranging from **NP** through to **ELEMENTARY**. By looking at the respective classes of program schemes where access to the numeric universe is removed, we move from the complexity class to its fragment of problems closed under extensions. In doing so, we show that many complexity-theoretic inequalities, *e.g.*, **NEXPTIME** $\neq$ **2-NEXPTIME**, and hypotheses, *e.g.*, **NEXPTIME** $\neq$ **EXPSPACE**, are equivalent to these inequalities and hypotheses for the fragments, *e.g.*, **NEXPTIME**$\cap\mathcal{EXT} \neq$ **2-NEXPTIME**$\cap\mathcal{EXT}$ and **NEXPTIME**$\cap\mathcal{EXT} \neq$ **EXPSPACE**$\cap\mathcal{EXT}$; and that these fragments can be logically captured (in the sense of Gurevich [8]: it is not immediately obvious that a semantically restricted complexity class such as **NEXPTIME**$\cap\mathcal{EXT}$ can be captured syntactically by a logic). This is interesting for the same reason that the equivalence results of Abiteboul *at al.* are interesting, *i.e.*, there exist computationally trivial problems not in **NEXPTIME**$\cap\mathcal{EXT}$ and **EXPSPACE**$\cap\mathcal{EXT}$. Whilst our equivalence results are similar to those of Abiteboul *et al.*, they involve different logics: recall, that there are problems in NPSQ(1)$\cap$**NP** not definable in $\mathcal{L}^{\omega}_{\infty\omega}$. Also, whereas the logics involved in the results due to Abiteboul *el al.* have a zero-one law (because they are fragments of $\mathcal{L}^{\omega}_{\infty\omega}$), our classes of program schemes define problems closed under extensions and so have a 'one-law' (apart from the program scheme which accepts no finite structures over some fixed signature). Not all proofs are included due to space limitations.

2 Program Schemes with Queues

Our signatures consist of a (non-empty) finite tuple of relation symbols and constant symbols. A *problem* is an isomorphism-closed class of finite structures over some fixed signature. If $\mathcal{A}$ and $\mathcal{B}$ are both finite σ-structures such that the domain of $\mathcal{A}$ is contained in the domain of $\mathcal{B}$, *i.e.*, $|\mathcal{A}| \subseteq |\mathcal{B}|$, and $\mathcal{B}$ restricted to $|\mathcal{A}|$ is $\mathcal{A}$ then $\mathcal{B}$ is an *extension* of $\mathcal{A}$ and we write $\mathcal{A} \subseteq \mathcal{B}$. If Ω is a problem, over the signature σ, for which: $\mathcal{A}$ and $\mathcal{B}$ are finite σ-structures; $\mathcal{A} \subseteq \mathcal{B}$; and $\mathcal{A} \in \Omega$, imply that $\mathcal{B} \in \Omega$ then Ω is said to be *closed under extensions*. The class of problems that are closed under extensions is $\mathcal{EXT}$. Our structures are always finite.

Program schemes are more 'computational' means for defining classes of problems than are logical formulae. A *program scheme* $\rho \in$ NPS(1) involves a finite set $\{x_1, x_2, \ldots, x_k\}$ of *variables* and is over a signature σ. It consists of a finite sequence of *instructions* where the first instruction is '`input(`x_1`,`x_2`,...,`x_m`)`', for some $m \leq k$, and the variables of $\{x_1, x_2, \ldots, x_m\}$ are known as the *input-output variables*, with the remaining variables being the *free variables*. The remaining instructions are one of the following: an *assignment instruction* of the form 'x_i `:=` y', where y is a variable or a constant symbol; a *guess instruction* of the form '`guess` x_i'; a *while instruction* of the form '`while` φ `do` $\alpha_1; \alpha_2; \ldots; \alpha_q$ `od`', where φ is a quantifier-free first-order formula over σ and where each of $\alpha_1, \alpha_2, \ldots, \alpha_q$ is another instruction of one of the forms given here; or an *accept instruction* `accept` or a *reject instruction* `reject`. A program scheme $\rho \in$ NPS(1)

over σ with $k-m$ free variables takes expansions $\mathcal{A}'$ of σ-structures with $k-m$ constants as input and computes on $\mathcal{A}'$ in the obvious way except that: execution of an instruction '`guess` x_i' non-deterministically assigns an element of $|\mathcal{A}'|$ to x_i; initially, every input-output variable is non-deterministically assigned a value from $|\mathcal{A}'|$; and if a computation encounters an accept or a reject instruction then the computation halts. Note that the value of a free variable never changes throughout a computation: the free variables appear as if they were constant symbols. The structure $\mathcal{A}'$ is *accepted* by ρ, and we write $\mathcal{A}' \models \rho$, iff there exists a computation of ρ such that an accept instruction is reached. We can easily build the usual 'if' and 'if-then-else' instructions using while instructions: henceforth, we shall assume that these instructions are at our disposal. Note that the class of structures accepted by a program scheme of NPS(1) is a problem.

Whereas a stack and arrays were incorporated into the program schemes of NPS(1) in [4,11,12,13], let us now augment these program schemes with access to a queue, *i.e.*, a first-in, first-out store. In addition to the above instructions, we include the instructions '`push` x_i' and 'x_i `:= pop`' although in this latter instruction we insist that x_i must be an input-output variable, whereas in the former it can also be a free variable. The first instruction takes the current value of the variable x_i and pushes this value onto the end of the queue (the value of x_i does not change and the length of the queue increases by 1), and the second takes the current value from the head of the queue and sets x_i to have this value (with the length of the queue decreasing by 1). Initially, the queue is empty and if ever we attempt to pop from an empty queue then that particular computation is deemed not to be accepting. We denote the program schemes of NPS(1) augmented with a queue by NPSQ(1).

We can also augment our program schemes with a numeric universe. That is, we can assume that the variables of a program scheme are of one of two types: variables of the first type, the *element type*, take values from the universe of the input structure (as they have done so far); and variables of the second type, the *numeric type*, take values from the *numeric universe*, namely $\{0, 1, \ldots, n-1\}$, where the input structure has size n (we assume that the universe of the input structure and the numeric universe are disjoint). We insist that all variables of numeric type must be input-output variables and that they are initialised to 0. Additionally, there are two constant symbols 0 and max which are always interpreted as the numbers 0 and $n-1$. The instructions available to the variables of numeric type, $t_1, t_2, \ldots, t_q$, say, are assignments of the form 't_i `:=` t_j', 't_i `:= 0`', 't_i `:=` max' and 't_i `:=` t_j `+ 1`' (if the variable t_j has value $n-1$ then execution of this instruction causes that particular computation to reject the input); and 't_i `=` t_j', 't_i `= 0`', 't_i `=` max' and their negations can appear as atoms and negated atoms in quantifier-free first-order formulae used as tests in while instructions (these formulae might be combinations of atoms involving variables of both element and numeric type). The class of program schemes NPSQ(1) augmented with a numeric universe is denoted $\text{NPSQ}_+(1)$. Again, the classes of structures accepted by the program schemes of $\text{NPSQ}_+(1)$ are problems.

Finite structures are abstract objects and as such generally have no canonical representation as a string over $\{0,1\}$, *i.e.*, as a bit-string. Nevertheless, we can still arrange for finite structures to be input to Turing machines by encoding them as strings (throughout, our Turing machines are non-deterministic). We say that a problem Ω over σ is *accepted* by a Turing machine M if M accepts exactly those bit-strings encoding structures in Ω; and so, in particular, as to whether an encoding $e_\sigma(\mathcal{A})$ of any σ-structure $\mathcal{A}$ is accepted by M is independent of the particular linear order of the elements chosen when encoding $\mathcal{A}$. We can easily choose our encoding scheme so that the length of $e_\sigma(\mathcal{A})$ is independent of the particular linear ordering chosen; and if $|\mathcal{A}| = n$ then we denote this length by $e_\sigma(n)$. Define $\mathcal{REC}$ as the class of problems accepted by some Turing machine. Note that not every Turing machine accepts a problem and that $\mathcal{REC}$ is not the class of recursive languages but the class of recursive problems. However, any (resp. recursive) language can be realised as a problem over the signature σ consisting of the binary relation symbol L and the unary relation symbol B (resp. accepted by a Turing machine). A string over $\{0,1\}$ can be considered as a σ-structure $\mathcal{A}$ (in fact, as a class of σ-structures) whose binary relation $L^{\mathcal{A}}$ describes a linear order, *i.e.*, the bit positions of the string, and whose unary relation $B^{\mathcal{A}}$ details which bits are 0 and which bits are 1.

Throughout, we identify a class of program schemes with the class of problems accepted by those program schemes (the same goes for sentences of logics and Turing machines).

Theorem 1. *A problem is accepted by a program scheme of* $NPSQ_+(1)$ *iff it can be accepted by a Turing machine*; *that is,* $NPSQ_+(1) = \mathcal{REC}$.

Proof. Suppose that the problem Ω is accepted by some (non-deterministic) Turing machine M. We shall construct a program scheme ρ of $\mathrm{NPSQ}_+(1)$ which simulates M. Without loss of generality, we can assume that: the Turing machine M has a two-way infinite work-tape and a read-only input-tape; the work-tape alphabet is $\{0,1,b\}$; and the input-tape has a left-hand marker symbol l and a right-hand marker symbol r which delimit the input string.

Essentially, the queue of our program scheme ρ will hold a description of the tapes of the Turing machine. We start with the work-tape. Let u and v be distinct elements of an input structure (we can guess these elements as the first act of our program scheme). In our description of the work-tape, we encode the work-tape symbol 0 as the triple (u,u,u), the work-tape symbol 1 as the triple (v,v,v) and the blank symbol b as the triple (u,u,v); and we use the triples (v,u,u) and (v,v,u) as delimiters. Let the work-tape at some instantaneous description (ID) of M be of the form $w_1, w_2, \ldots, w_i, \ldots, w_m$, reading from left to right where w_1 (resp. w_m) is the first blank symbol to the left (resp. right) of the work-tape head beyond which the work-tape is entirely blank. Furthermore, suppose that the work-tape head is scanning the symbol w_i. The queue will consist of: the triple encoding w_1; the triple encoding w_2; ...; the triple (v,u,u) (to denote that the head is scanning the next symbol); the triple encoding w_i; ...; the triple encoding w_m; and the triple (v,v,u) (to denote that we have reached the end of our description of the work-tape). We shall describe the state

of the Turing machine M in any ID using a constant number of variables of our program scheme ρ. Also, should our program scheme ρ know which move of the Turing machine M to simulate and the effect upon the work-tape and the work-tape head, we could easily simulate this move by making appropriate manipulations of the queue.

Consequently, we are left with the problem of dealing with the input tape. This is more complicated as initially our program scheme is not given an encoding of the input structure $\mathcal{A}$, which has size n, say: it simply has access to the raw finite structure itself. However, we can build our own encoding of $\mathcal{A}$. Prior to any simulation of the Turing machine M, our program scheme guesses a linear ordering on $|\mathcal{A}|$ and stores this linear ordering on the queue. Note that because we have access to the (disjoint) domain $\{0, 1, \ldots, n-1\}$, the capacity to count using the elements of this domain and the integers 0 and $n-1$ (through the constants 0 and max), we can ensure that our guessed linear order contains every element of $|\mathcal{A}|$ exactly once.

Suppose that our guessed linear order is $u_0, u_1, \ldots, u_{n-1}$. This linear order defines a concrete encoding of our input structure and it is this encoding which we assume is presented to the Turing machine M. We use our linear order to write a description of the input-tape of M on our queue.

It is now straightforward, given descriptions of the input-tape and the work-tape on the queue, to simulate the computation of M. The converse, that any problem in $\text{NPSQ}_+(1)$ is in $\mathcal{REC}$, is trivial. □

The question remains as to what sort of problems can be accepted by program schemes of NPSQ(1).

Theorem 2. *A problem is accepted by a program scheme of NPSQ*(1) *iff it can be accepted by a Turing machine and is closed under extensions*; *that is,* $NPSQ(1) = \mathcal{REC} \cap \mathcal{EXT}$.

Proof. Let $\mathcal{A}$ and $\mathcal{B}$ be σ-structures such that $\mathcal{A} \subseteq \mathcal{B}$. If $\mathcal{A}$ is accepted by some program scheme ρ of NPSQ(1) then we can mirror an accepting computation in a computation of ρ on $\mathcal{B}$ (essentially, because our tests in while instructions are quantifier-free). Hence, $\text{NPSQ}(1) \subseteq \mathcal{REC} \cap \mathcal{EXT}$.

Conversely, suppose that Ω is a problem, over the signature σ, in $\mathcal{REC} \cap \mathcal{EXT}$. In particular, Ω is accepted by some Turing machine M. Construct a program scheme ρ' of NPSQ(1) as in the proof of Theorem 1 except that when the linear order is guessed, the checks made are that every element appears in the linear order at most once and that every constant appears in the linear order. Whenever ρ' makes a guess, include code to ensure that the guess is always an element in the domain of the linear order.

Suppose that the σ-structure $\mathcal{A}$ is in Ω; that is, every encoding of $\mathcal{A}$ is accepted by M. Then there is a computation of ρ' on input $\mathcal{A}$ which guesses a linear order whose domain is the whole of $|\mathcal{A}|$; and as a result, this computation of ρ' on input $\mathcal{A}$ simulates that of M on the respective encoding of $\mathcal{A}$. Thus, $\mathcal{A}$ is accepted by ρ'.

Suppose that the σ-structure $\mathcal{A}$ is accepted by ρ'. Then there is an accepting computation where this computation guesses a linear order whose elements come from the subset B of $|\mathcal{A}|$. Let $\mathcal{B}$ be the restriction of $\mathcal{A}$ to B (and so $\mathcal{A}$ is an extension of $\mathcal{B}$). The accepting computation of ρ' on input $\mathcal{A}$ simulates an accepting computation of M on input the respective encoding of $\mathcal{B}$ (because any guess results in an element of B). Hence, this encoding of $\mathcal{B}$ is accepted by M and thus $\mathcal{B} \in \Omega$. But Ω is closed under extensions and so $\mathcal{A} \in \Omega$. The result follows. □

3 Zero-One Laws

We can extend our class of program schemes NPSQ(1) to a hierarchy of classes of program schemes essentially by interleaving applications of universal quantifiers with the basic constructs of the program schemes of NPSQ(1). In more detail, assume that we have defined a class of program schemes NPSQ($2m-1$), for some $m \geq 1$, and that any program scheme has associated with it: a set of input-output variables; a set of free variables; and a set of bound variables.

Definition 1. *Let the program scheme $\rho \in NPSQ(2m-1)$ be over the signature σ, and suppose that ρ has: input-output variables $x_1, x_2, \ldots, x_k$; free variables $x_{k+1}, x_{k+2}, \ldots, x_{k+s}$; and bound variables $x_{k+s+1}, x_{k+s+2}, \ldots, x_{k+s+t}$. Let $x_{i_1}, x_{i_2}, \ldots, x_{i_p}$ be free variables of ρ. Then $\forall x_{i_1} \forall x_{i_2} \ldots \forall x_{i_p} \rho$ is a program scheme of NPSQ(2m), which we denote by ρ', with: no input-output variables; free variables those of $\{x_{k+1}, x_{k+2}, \ldots, x_{k+s}\} \setminus \{x_{i_1}, x_{i_2}, \ldots, x_{i_p}\}$; and the remaining variables of $\{x_1, x_2, \ldots, x_{k+s+t}\}$ as its bound variables.*

A program scheme such as ρ' takes expansions $\mathcal{A}'$ of σ-structures $\mathcal{A}$ by adjoining $s-p$ constants as input (one for each free variable), and ρ' accepts such an expansion $\mathcal{A}'$ iff for every expansion $\mathcal{A}''$ of $\mathcal{A}'$ by p additional constants (one for each variable x_{i_j}, for $j \in \{1, 2, \ldots, p\}$), $\mathcal{A}'' \models \rho$ (the computation on such an expansion $\mathcal{A}''$ always starts with the queue empty). □

Definition 2. *A program scheme $\rho' \in NPSQ(2m-1)$, for some $m \geq 2$, over the signature σ, is defined exactly as is a program scheme of NPSQ(1) except that the test in any while instruction is a program scheme $\rho \in NPSQ(2m-2)$. The bound variables of ρ' consist of the bound variables of any test in any while instruction; all free variables in any test in any while instruction are input-output or free variables of ρ'; and there may be other free and input-output variables (appearing in ρ' at the 'top level' but not in any test). Of course, any free variable never appears on the left-hand side of an assignment instruction, in a guess instruction or in a pop instruction in ρ'.*

Suppose that a program scheme $\rho' \in NPSQ(2m-1)$ has s free variables. Then it takes expansions $\mathcal{A}'$ of σ-structures $\mathcal{A}$ by adjoining s constants as input and computes on $\mathcal{A}'$ in the obvious way; except that when some while instruction is encountered, the test, a program scheme $\rho \in NPSQ\ (2m-2)$, is evaluated according to the expansion of $\mathcal{A}'$ by the current values of any relevant input-output variables of ρ'. In order to evaluate this test, the queue associated with ρ

is initialised as empty and when the test has been evaluated the computation of ρ' resumes accordingly with its queue and input-output and free variables being as they were immediately prior to evaluation. □

So, we obtain a hierarchy NPSQ(1) $\subseteq$ NPSQ(2) $\subseteq \ldots \subseteq \cup\{\text{NPSQ}(i) : i = 1, 2, \ldots\} = \text{NPSQ}$. Similar hierarchies have been defined previously in [4,11,13] but based around stacks and arrays and not queues. We omit the proofs of the following two results but remark that the proof of the latter result follows from the proof of the former.

Theorem 3. *The class of program schemes NPSQ has a zero-one law.* □

Then the logic formed by extending first-order logic FO with a *vectorized sequence of Lindström quantifiers* corresponding to the problem Ω is denoted $(\pm\Omega)^*$[FO]: see [6] for definitions. Such a logic is a *vectorized Lindström logic*.

Corollary 1. $NPSQ = \bigcup\{(\pm\Omega)^*[FO] : \Omega \in \mathcal{REC} \cap \mathcal{EXT}\}$. □

Essentially, Corollary 1 equates the 'syntactically-defined' class of problems NPSQ with the 'semantically-defined' class of problems definable using first-order constructs in tandem with 'recursive extension-closed' operators. Whilst NPSQ is a natural sub-class of a computationally complete class of program schemes, there are problems in $\mathcal{REC}$ which have a zero-one law but which are not in NPSQ, one such being the 'Hamiltonian cycle' problem.

4 A Simple Application

It is a question of great importance in finite model theory and database theory as to whether there is a logic which captures exactly the class of polynomial-time solvable problems. Of course, one has to be precise about what one means by a logic and Gurevich [8] has formulated a definition which has been widely adopted. A *logic* L is given by a pair of functions (Sen, Sat) satisfying the following conditions. The function Sen associates with every signature σ a recursive set $Sen(\sigma)$ whose elements are called *L-sentences over σ*. The function Sat associates with every signature a recursive relation $Sat_\sigma(\mathcal{A}, \varphi)$, where $\mathcal{A}$ is a σ-structure and φ is a sentence of L. We say that $\mathcal{A}$ *satisfies* φ if $Sat_\sigma(\mathcal{A}, \varphi)$ holds. Furthermore, we require that $Sat_\sigma(\mathcal{A}, \varphi)$ iff $Sat_\sigma(\mathcal{B}, \varphi)$ when $\mathcal{A}$ and $\mathcal{B}$ are isomorphic σ-structures. As yet no-one has exhibited a logic (in Gurevich's sense) to capture any mainstream complexity class (for which the expectation is that it is properly) contained in **NP**. Logical characterizations of complexity classes on the class of ordered finite structures usually suffer from the difficulty that deciding whether a formula of the logic is 'order invariant' is undecidable. It is straightforward to verify that the classes of program schemes NPSQ (together with the classes therein) and NPSQ$_+$(1) are logics in the sense of Gurevich; and so, for example, Theorem 2 implies that the class of recursively-solvable problems that are closed under extensions can be logically captured. However,

by imposing resource conditions upon our program schemes, we can logically capture other classes of problems.

A problem Ω, over some signature σ, is accepted by a non-deterministic Turing machine M *in time* $f(x)$ (resp. *using space* $g(x)$) if M accepts Ω and for every encoding $e_\sigma(\mathcal{A})$ of any σ-structure $\mathcal{A}$ of size n in Ω, there is an accepting computation of M on input $e_\sigma(\mathcal{A})$ taking time at most $f(e_\sigma(n))$ (resp. using space at most $g(e_\sigma(n))$). In contrast, the time and space complexities of a program scheme of $\mathrm{NPSQ}_+(1)$ or $\mathrm{NPSQ}(1)$ on some input structure are measured in terms of the size of the input structure. In order to evaluate the time taken or the space used in some computation of a program scheme of $\mathrm{NPSQ}_+(1)$ or $\mathrm{NPSQ}(1)$ on some input structure of size n, we treat the execution of every assignment, quantifier-free test, pop, push, etc., as taking one unit of time, and we regard a variable or a place on the queue as occupying one unit of space. As for a non-deterministic Turing machine, the complexity of a program scheme of $\mathrm{NPSQ}_+(1)$ or $\mathrm{NPSQ}(1)$ is derived using accepting computations only.

For any function $f(x)$, let $\mathcal{POLY}(f(x))$ denote $\{p_0(x).(f(q_0(x)))^k + p_1(x).(f(q_1(x)))^{k-1} + \ldots + p_{k-1}(x).f(q_{k-1}(x)) + p_k(x) : k \geq 0, p_0, p_1, \ldots, p_k, q_0, q_1, \ldots, q_{k-1}$ are polynomials with integer coefficients$\}$. If $\mathcal{C}$ is a class of functions then $\mathcal{POLY}(\mathcal{C}) = \bigcup_{f(x)\in\mathcal{C}} \mathcal{POLY}(f(x))$. Denote the class of non-deterministic Turing machines that accept problems in time (resp. using space) $f(x)$ where $f(x)$ is a function from the class of functions $\mathcal{C}$ as $\mathrm{NTM}^t(\mathcal{C})$ (resp. $\mathrm{NTM}^s(\mathcal{C})$), with $\mathrm{NPSQ}^t(1)(\mathcal{C})$ and $\mathrm{NPSQ}^t_+(1)(\mathcal{C})$ (resp. $\mathrm{NPSQ}^s(1)(\mathcal{C})$ and $\mathrm{NPSQ}^s_+(1)(\mathcal{C})$) defined likewise. The proofs of Theorems 1 and 2 yield the following.

Corollary 2. *For any class of functions* $\mathcal{C}$:

- $NTM^t(\mathcal{POLY}(\mathcal{C})) = NPSQ^t_+(1)(\mathcal{POLY}(\mathcal{C}))$
- $NTM^t(\mathcal{POLY}(\mathcal{C})) \cap \mathcal{EXT} = NPSQ^t(1)(\mathcal{POLY}(\mathcal{C}))$
- $NTM^s(\mathcal{POLY}(\mathcal{C})) = NPSQ^s_+(1)(\mathcal{POLY}(\mathcal{C}))$
- $NTM^s(\mathcal{POLY}(\mathcal{C})) \cap \mathcal{EXT} = NPSQ^s(1)(\mathcal{POLY}(\mathcal{C}))$. □

We can regard $\mathrm{NPSQ}^t(1)(\mathcal{POLY}(\mathcal{C}))$, for example, as a logic (in Gurevich's sense) by identifying program schemes from this class with pairs $(\rho, F(x))$, where ρ is a program scheme of $\mathrm{NPSQ}(1)$ and $F(x)$ is a function from $\mathcal{POLY}(\mathcal{C})$. Of course, this requires that the functions of $\mathcal{C}$ form a recursive set and that there is a Turing machine which when given a function $f(x)$ from $\mathcal{C}$ and a positive integer n, computes the value $f(n)$. Thus, Corollary 2 can be used to logically capture a whole host of complexity classes including **NP**, **PSPACE**, **NEXPTIME**, **EXPSPACE**, **2-NEXPTIME**, **2-EXPSPACE**, ..., **ELEMENTARY**. However, Corollary 2 can also be used to logically characterize 'fragments' of (the above) complexity classes obtained by intersecting the complexity class with the class of problems $\mathcal{EXT}$. Note that this is a 'semantic' rather than a 'syntactic' restriction and so it is not immediately obvious as to whether such a fragment of a complexity class can be logically captured. Furthermore, existing complexity-theoretic inequalities or hypotheses can be used to derive analogous ones for the respective fragments of the complexity classes in question.

Applying Corollary 2, we know that **NEXPTIME** $\neq$ **2-NEXPTIME**. Corollary 2 implies that **NEXPTIME** $\cap \mathcal{EXT} \neq$ **2-NEXPTIME** $\cap \mathcal{EXT}$ and that the class of problems accepted by program schemes of NPSQ(1) restricted to run in time $2^{2^{p(x)}}$, for some polynomial $p(x)$, properly contains the class of problems accepted by program schemes of NPSQ(1) restricted to run in time $2^{p(x)}$, for some polynomial $p(x)$. Also, it is open as to whether **NEXPTIME** = **EXPSPACE**, which, by Corollary 2, is equivalent to whether **NEXPTIME** $\cap$ $\mathcal{EXT}$ = **EXPSPACE** $\cap \mathcal{EXT}$ and to whether the class of problems accepted by program schemes of NPSQ(1) restricted to run in time $2^{p(x)}$, for some polynomial $p(x)$, is the same as the class of problems accepted by program schemes of NPSQ(1) restricted to run using space $2^{p(x)}$, for some polynomial $p(x)$.

We feel that it is interesting that open complexity-theoretic questions are equivalent to analogous logical questions involving classes of problems with zero-one laws, and even closed under extensions. Of course, these latter classes of problems do not contain some computationally-trivial problems, yet still they have a definitive role to play in the question of whether two complexity classes, such as **NEXPTIME** and **EXPSPACE** which have significant complexity-theoretic capacities, are identical.

References

1. S. Abiteboul, M.Y. Vardi and V. Vianu, Fixpoint logics, relational machines and computational complexity, *J. ACM* **44** (1997) 30–56.
2. S. Abiteboul and V. Vianu, Generic computation and its complexity, *Proc. 23rd Ann. ACM Symp. Theory Comput.*, ACM Press (1991) 209–219.
3. S. Abiteboul and V. Vianu, Computing with first-order logic, *J. Comput. System Sci.* **50** (1995) 309–335.
4. A.A. Arratia-Quesada, S.R. Chauhan and I.A. Stewart, Hierarchies in classes of program schemes, *J. Logic Computat.* **9** (1999) 915–957.
5. A. Dawar, A restricted second-order logic for finite structures, *Informat. Computat.* **143** (1998) 154–174.
6. H.D. Ebbinghaus and J. Flum, *Finite Model Theory*, Springer-Verlag (1995).
7. R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: *Complexity of Computation* (ed. R.M. Karp), Volume 7 of SIAM-AMS Proceedings (1974) 43–73.
8. Y. Gurevich, Logic and the challenge of computer science, in: *Current Trends in Theoretical Computer Science* (ed. E. Börger), Computer Science Press (1988) 1–57.
9. Y. Gurevich, Evolving algebras 1993: Lipari guide, in: *Specification and Validation* (ed. E. Börger), Oxford University Press (1995) 9–36.
10. N. Immerman, *Descriptive Complexity*, Springer-Verlag (1998).
11. I.A. Stewart, Program schemes, arrays, Lindström quantifiers and zero-one laws, *Lecture Notes in Computer Science Volume* 1683, Springer-Verlag (1999) 374–388 (full version to appear in *Theoret. Comput. Sci.*).
12. I.A. Stewart, Using program schemes to logically capture polynomial-time on certain classes of structures, *submitted for publication*.
13. I.A. Stewart, Logical definability versus computational complexity: another equivalence, *submitted for publication*.

Algebraic Properties for P-Selectivity

Lane A. Hemaspaandra[1,⋆], Harald Hempel[2,⋆⋆], and Arfst Nickelsen[3]

[1] Department of Computer Science, University of Rochester, Rochester, NY 14627-0226, USA, lane@cs.rochester.edu
[2] Institut für Informatik, Friedrich-Schiller-Universität Jena, D-07743 Jena, Germany, hempel@informatik.uni-jena.de
[3] Fachbereich Informatik, Technische Universität Berlin, D-10587 Berlin, Germany, nicke@cs.tu-berlin.de

Abstract. Karp and Lipton, in their seminal 1980 paper, introduced the notion of advice (nonuniform) complexity, which since has been of central importance in complexity theory. Nonetheless, much remains unknown about the optimal advice complexity of classes having polynomial advice complexity.
In particular, let P-sel denote the class of all P-selective sets [23] For the *nondeterministic* advice complexity of P-sel, linear upper and lower bounds are known [10]. However, for the *deterministic* advice complexity of P-sel, the best known upper bound is quadratic [13], and the best known lower bound is the linear lower bound inherited from the nondeterministic case. This paper establishes an algebraic sufficient condition for P-sel to have a linear upper bound: If all P-selective sets are associatively P-selective then the deterministic advice complexity of P-sel is linear. (The weakest previously known sufficient condition was P = NP.) Relatedly, we prove that every associatively P-selective set is commutatively, associatively P-selective.

1 Introduction

Selman [23,24,25,26] defined the P-selective sets about twenty years ago. In addition to being of interest in their own right, they have recently had some surprising applications. For example, selectivity is a powerful tool in the study of search versus decision problems [4], and nondeterministic generalizations of selectivity are the key tools used to show that even NP machines cannot uniquely refine satisfying assignments unless the polynomial hierarchy collapses [8], that even weaker refinements are also precluded unless the polynomial hierarchy collapses [21,18], and that many cardinality types of nondeterministic function classes cannot collapse unless the polynomial hierarchy collapses [9].

⋆ Supported in part by grants NSF-CCR-9322513 and NSF-INT-9815095/DAAD-315-PPP-gü-ab.
⋆⋆ Supported in part by grant NSF-INT-9815095/DAAD-315-PPP-gü-ab. Work done while visiting the University of Rochester under a NATO Postdoctoral Science Fellowship from the DAAD's "Gemeinsames Hochschulsonderprogramm III von Bund und Ländern" program.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 49–58, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Definition 1 ([23]). *A set B is* P*-selective if and only if there is a (total) polynomial-time function $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ such that*

1. $(\forall x, y)[f(x, y) = x \vee f(x, y) = y]$.
2. $(\forall x, y)[(x \in B \vee y \in B) \implies f(x, y) \in B]$.

We call such an f a P*-selector function for B.*

That is, a set B is P-selective if there is a polynomial-time function f that given any two strings always chooses one of them, and f does this in such a way that if exactly one of the strings is in B then the P-selector function chooses that string. $f(x, y)$ is often described, very informally, as choosing one of x or y that is "more likely (actually, no less likely) to be in the set."

The P-selective sets have been extensively studied, and much about them is well understood. Though some P-selective sets are very complex—highly undecidable—the P-selective sets nonetheless have a broad range of structural simplicity properties. Most crucially in terms of the study in this paper, Ko [13] showed that they have low nonuniform complexity (P-sel $\subseteq$ P/$\mathcal{O}(n^2)$). A few among the many other simplicity results known to hold are: Ko and Schöning [15] showed that all P-selective sets in NP are in the second level of the low hierarchy of Schöning [22], and Allender and Hemachandra [1] showed that the Ko-Schöning result is the strongest lowness result for P-sel that holds with respect to all oracles; a long line of work starting with Selman [23] and Toda [28] has shown that no P-selective set can be NP-hard under $\leq^{\mathrm{p}}_m$ or various other reductions unless P = NP; Naik and Selman [19] have shown that no P-selective set can be truth-table-hard for NP unless certain (intuitively unlikely) containments hold in the relationship between adaptive and nonadaptive queries to NP; and as a consequence of the work of Ko [13] and Köbler and Watanabe [16], no P-selective set can be truth-table-hard for NP unless the polynomial hierarchy collapses to $\mathrm{ZPP}^{\mathrm{NP}}$, where ZPP as usual denotes expected polynomial time. Though much is known about the P-selective sets and their refinements and generalizations, including quite current work (e.g., [20]), much remains open.

In this paper, we show that P-selector functions with nice algebraic properties have simplicity properties far beyond those known to hold for general P-selective sets. More generally, we study the class of languages one obtains from P-selector functions having algebraic properties, and the possibility of obtaining such algebraically nice P-selector functions.

In particular, Section 3 shows that any P-selective set having an associative P-selector function also has a commutative, associative P-selector function. It follows that sets having an associative P-selector can be accepted by deterministic advice interpreters using only a linear amount of advice. In contrast, the best upper bound on the deterministic advice complexity of the P-selective sets is the quadratic bound obtained by Ko [13].

Our result provides a new sufficient condition—all P*-selective sets are associatively* P*-selective—for all* P*-selective sets having linear deterministic advice; the weakest previously known sufficient condition was the extremely demanding assumption that* P = NP.

Section 4 shows that associatively P-selective sets with weak census functions can never be coNP-immune. Section 5 establishes a structural sufficient condition for all P-selective sets being associatively, commutatively P-selective.

2 Definitions

A binary function f is *associative* if and only if, for all a, b, and c, it holds that $f(a, f(b, c)) = f(f(a, b), c)$. A binary function f is *associative at each length* if, for all a, b, and c satisfying $|a| = |b| = |c|$, it holds that $f(a, f(b, c)) = f(f(a, b), c)$. A binary function f is *commutative* if and only if, for all a and b, $f(a, b) = f(b, a)$. We will use the following notational shorthands for the classes that we will study. (The ordering of "A" and "C" changes to avoid confusion with the existing class "AC," and to make clear that the "ℓ" modifies just the "A.")

Definition 2. *1.* P-sel = $\{B \mid B$ *is* P*-selective*$\}$.
2. A-sel = $\{B \mid B$ *is* P*-selective via an associative* P*-selector function*$\}$.
3. C-sel = $\{B \mid B$ *is* P*-selective via a commutative* P*-selector function*$\}$.
4. CA-sel = $\{B \mid B$ *is* P*-selective via a commutative, associative* P*-selector function*$\}$.
5. A_ℓ-sel = $\{B \mid B$ *is* P*-selective via a* P*-selector function that is associative at each length*$\}$.
6. A_ℓC-sel = $\{B \mid B$ *is* P*-selective via a commutative* P*-selector function that is associative at each length*$\}$.

From these definitions, CA-sel $\subseteq$ A-sel $\subseteq$ A_ℓ-sel $\subseteq$ P-sel, CA-sel $\subseteq$ A_ℓC-sel $\subseteq$ A_ℓ-sel $\subseteq$ P-sel, and A_ℓC-sel $\subseteq$ C-sel $\subseteq$ P-sel. The following fact is well-known.

Fact 1 P-sel = C-sel.

The fact holds as if B is P-selective via P-selector function f, then $f'(x, y) = \max(f(x, y), f(y, x))$ is a commutative function that remains a P-selector for B.

Advice classes capture the information content of sets with respect to some class of decoder (usually P).

Definition 3 ([12]).

1. Let $\mathcal{C}$ be any complexity class and f be any function mapping from $\mathbb{N}$ to $\mathbb{N}$. A set B is in $\mathcal{C}/f(n)$ if and only if there exist a set $C \in \mathcal{C}$ and a function $h : 1^* \to \Sigma^*$ such that for all $x \in \Sigma^*$,
(a) $|h(1^{|x|})| = f(|x|)$, and
(b) $x \in B \iff \langle x, h(1^{|x|})\rangle \in C$.
2. For any complexity class $\mathcal{C}$ and any function class $\mathcal{F}$, $\mathcal{C}/\mathcal{F} = \bigcup_{f \in \mathcal{F}} \mathcal{C}/f(n)$.

In this paper, we will be particularly interested in P/$\mathcal{O}(n^2)$, P/$\mathcal{O}(n)$, P/n+1, and P/n.

Let $\mathcal{F}$ be any function complexity class. A set B is $\mathcal{F}$-printable if there is a function $g \in \mathcal{F}$ such that, for all x, $g(x)$ outputs (in some fixed, standard

way of encoding sets as strings) $B \cap (\Sigma^0 \cup \Sigma^1 \cup \cdots \cup \Sigma^{|x|})$ [3]. That is, there is a function from $\mathcal{F}$ that finds all elements in the set up to a given length. By tradition, P-printable denotes the FP-printable sets.

As is standard, FP is the class of all (total) polynomial-time computable functions. $\mathrm{FP}^{\mathrm{NP}}$ denotes the class of all functions that are computable in polynomial-time with the help of an NP oracle.

3 Associativity Drops Deterministic Advice Complexity to Linear

Are all associatively P-selective sets commutatively, associatively P-selective? If they are, this will not only collapse two of our classes, but also will allow us to use results from tournament theory in our study of associative selector functions (since, as we will discuss later, commutative selector functions have a close relationship to tournaments). Unfortunately, Fact 1 does *not* say that A-sel = CA-sel. The reason it does not guarantee this is that it is conceptually possible that the transformation from f to f' that is stated just after Fact 1 will, while gaining commutativity, destroy associativity. The following theorem, our main result, states that this is never the case. (All proofs here omitted due to space can be found in [6].)

Theorem 2. A-sel = CA-sel.

Theorem 3. A_ℓ-sel = A_ℓC-sel.

Ko showed the following result.

Theorem 4 ([13]). P-sel $\subseteq \mathrm{P}/\mathcal{O}(n^2)$.

That is, deterministic advice interpreters given quadratic advice accept the P-selective sets. It is also known that nondeterministic advice interpreters with linear advice accept the P-selective sets, and this is optimal.

Theorem 5 ([10]). P-sel $\subseteq \mathrm{NP}/n{+}1$ *but* P-sel $\not\subseteq \mathrm{NP}/n$.

It is natural to wonder whether those results can be unified into the very strong claim: P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$. Currently, no proof of this claim is known and people doing research on selectivity generally doubt that it holds. However, proving the claim false seems unlikely in the immediate future, as by Theorem 5 any such proof would implicitly prove P $\neq$ NP.

Nonetheless, the following result shows that all associatively P-selective sets *are* accepted by deterministic advice interpreters with linear advice.

Theorem 6. A_ℓ-sel $\subsetneq \mathrm{P}/n{+}1$.

Corollary 1. A-sel $\subsetneq \mathrm{P}/n{+}1$.

Proof of Theorem 6: Let $B \in \mathrm{A}_\ell$-sel. Let f be a commutative P-selector for B that is associative at each length. By Theorem 3 we know that such a commutative P-selector for B must exist.

Let $n \in \mathbb{N}$. Consider the directed graph $G_n = (B^{=n}, E_n)$, where $E_n = \{(x,y) \mid x,y \in B^{=n} \wedge x \neq y \wedge f(x,y) = y\}$. G_n is a tournament—a directed graph such that between any two nodes u, v, $u \neq v$, there is exactly one of the edges (u,v) or (v,u).

E_n is a transitive relation. To see this, let $x,y,z \in B^{=n}$. Suppose $(x,y) \in E_n$ and $(y,z) \in E_n$. Note that this implies $x \neq y$, $y \neq z$, and $x \neq z$. It also implies that $f(x,y) = y$ and $f(y,z) = z$. Thus, $f(f(x,y),z) = z$ and $f(x,f(y,z)) = f(x,z)$. Since f is associative at each length we have $f(f(x,y),z) = f(x,f(y,z))$ and so $f(x,z) = z$, which implies $(x,z) \in E_n$. Thus, E_n is transitive.

Tournaments such that their edge set is a transitive relation are called transitive tournaments. We will use the following result (which can be viewed as a tournament-language expression of the fact that each finite linear order has a minimal element).

Proposition 1 ([17]). *The following statements are equivalent for tournaments $G = (V,E)$:*

1. *G is a transitive tournament.*
2. *G contains no directed cycles.*
3. *Every induced (directed) subgraph $G' = (V',E')$, $V' \neq \emptyset$, of G contains a (distance-one) source node, i.e., a node $u \in V'$ such that for all $v \in V'$, $v \neq u$, $(u,v) \in E'$.*

Since f is associative at each length, we have for all $n \in \mathbb{N}$ that G_n is a transitive tournament, which in turn by Proposition 1 implies that (if G_n is not the empty tournament) G_n contains a source node s_n. In other words, for all $n \in \mathbb{N}$, if $B^{=n} \neq \emptyset$ then there exists a string $s_n \in B^{=n}$ such that for all $x \in B^{=n}$, $f(x,s_n) = x$. Since $s_n \in B^{=n}$, clearly for each length n string $z \in \overline{B}$ we have $f(z,s_n) = s_n$.

Define, for all $n \in \mathbb{N}$, $h(1^n) = 1s_n$ if $B^{=n} \neq \emptyset$, and $h(1^n) = 0^{n+1}$ otherwise. Note that h on any input 1^n outputs a string of length exactly $n+1$. Let $L = \{\langle x, 1y\rangle \mid |x| = |y| \wedge f(x,y) = x\}$. Clearly, $L \in \mathrm{P}$. Due to the above construction we also have, for all $x \in \Sigma^*$,

$$x \in B \iff \langle x, h(1^{|x|})\rangle \in L.$$

This shows that $B \in \mathrm{P}/n{+}1$.

It remains to show that $\mathrm{P}/n{+}1 - \mathrm{A}_\ell\text{-sel} \neq \emptyset$. In fact, it clearly holds even that $\mathrm{P}/1 - \text{P-sel} \neq \emptyset$. The proof of this claim consists of a diagonalization against all P-selectors. Let $f_1, f_2, \ldots$ be an enumeration (not necessarily effective, though in fact there do exist such effective enumerations) of all polynomial-time selector functions (i.e., all polynomial-time functions such that for all x and y it holds that $f(x,y) \in \{x,y\}$). Set $B = \{0^{2i-1} \mid f_i(0^{2i-2}, 0^{2i-1}) = 0^{2i-2}\} \cup \{0^{2i-2} \mid f_i(0^{2i-2}, 0^{2i-1}) = 0^{2i-1}\}$. B is not P-selective, yet $B \in \mathrm{P}/1$. ❑

Regarding the previous result, note that, more generally, if a set A and a (not necessarily commutative) P-selector f for it have the property that at each nonempty length m there is some string y in the set at that length—or even at some length linearly related to that length—such that $f(y,z) = z$ or $f(z,y) = z$ for each z of length m, then A is in $\mathrm{P}/\mathcal{O}(n)$. However, among ways of ensuring that this condition is met, we feel that associativity is a particularly natural and well-structured property to study.

The number of advice bits in Theorem 6 and Corollary 1, $n+1$, is optimal.

Theorem 7. A-sel $\not\subseteq \mathrm{P}/n$.

Corollary 2. A_ℓ-sel $\not\subseteq \mathrm{P}/n$.

In fact, for any recursive function f, A-sel $\not\subseteq \mathrm{DTIME}[f(n)]/n$. The reason for this is that we may take the P-selective set that Hemaspaandra and Torenvliet [10] prove is not in $\mathrm{DTIME}[f(n)]/n$ and, just as in the proof of Theorem 7, may complete its P-selector function in a manner that achieves associativity.

We mention in passing that—since $\log_2((2^{n+1}-1)+1) = n+1$ and thus one can include in the advice-seeking tournament all lengths up to the current one—the $n+1$ advice bounds in Theorem 6 and Corollary 1 hold even in the "strong advice" model of Ko, Balcázar, and Schöning [14,2] (see also the discussion in [10]).

We note that our results relativize. In particular, one can relativize them by any particular NP∩coNP set, and so regarding associativity and the NP-selective sets (equivalently the NPSV_t-selective sets and equivalently the $\mathrm{FP}^{\mathrm{NP}\cap\mathrm{coNP}}$-selective sets, see [8,7] for definitions and discussion), it follows from our results that all length-associatively NP-selective sets are in $(\mathrm{NP}\cap\mathrm{coNP})/n+1$. In fact, essentially all of this paper's results can be stated and proven more generally so as to apply even to a variety of nondeterministic selectivity classes; see [5] for full details.

4 Printability and Nonimmunity

Associativity yields additional simplicity properties. Let us consider nonimmunity results (i.e., presence of infinite subsets). For general P-selective sets, Theorem 8 holds. In contrast, for associatively selective sets, we have Theorem 9. To see how these bounds relate, note that $\mathrm{UP}^{\mathrm{NP}} \subseteq \mathrm{NP}^{\mathrm{NP}}$.

Theorem 8. *Every infinite* P*-selective set B has an infinite* $\mathrm{FP}^{B\oplus\mathrm{NP}^{\mathrm{NP}}}$*-printable subset.*

Theorem 9. *Every infinite* A*-selective (or even* A_ℓ*-selective) set B has an infinite* $\mathrm{FP}^{B\oplus\mathrm{UP}^{\mathrm{NP}}}$*-printable subset.*

Theorem 8 should be contrasted with the result of Hemaspaandra, Ogihara, Zaki, and Zimand (personal communication, 1999) that P-sel is not (weak-$\mathrm{FP}^{\mathrm{NP}^{\mathrm{NP}}}$-rankable)-immune, a result related to the study in [11].

Proof of Theorem 9: Let B be an infinite A_ℓ-selective set. Let f be a commutative P-selector of B that is associative at each length. By Theorem 3 we know that such a selector for B exists.

Consider the function *score* that is defined as $score(x) = \|\{z \in \Sigma^{|x|} \mid f(x,z) = x\}\|$. Note that for each $n \in \mathbb{N}$ and for each $x \in \Sigma^n$, (a) $x \notin B \implies score(x) \leq 2^n - \|B^{=n}\|$ and (b) $x \in B \implies score(x) \geq 2^n - \|B^{=n}\| + 1$. So for all $x \in B$ and all $y \notin B$ such that $|x| = |y|$, $score(y) < score(x)$. It follows from the associativity-at-each-length of f that for all n, $\max\{score(y) \mid |y| = n\} = 2^n$ (see also Proposition 1 and the comment immediately after it, applying both to the tournament on Σ^n defined by the associative-at-each-length selector function f). So for each n, there exists exactly one string at length n, call it d_n, at length n, such that $score(d_n) = \max\{score(y) \mid |y| = n\} = 2^n$. Thus, the set

$$F = \{\langle 1^n, z\rangle \mid n \in \mathbb{N} \wedge z \in \Sigma^* \wedge (\exists \omega : |\omega| = n - |z|)[score(z\omega) = 2^n]\}$$

is in $\mathrm{UP}^{\mathrm{NP}}$. So, given 1^n, computing d_n can be done in polynomial time with the help of the oracle F. Let $C = B \cap \{d_i \mid i \geq 0\}$. Clearly, as B is infinite, and C is both infinite and $\mathrm{FP}^{B \oplus \mathrm{UP}^{\mathrm{NP}}}$-printable. ❑

If one cares about the number of queries to B needed in the printability claims of Theorems 8 and 9 the following can be observed. As communicated to us by Till Tantau [27], one can limit one's queries to B to a logarithmic number via putting the queries into a Toda-like ordering and binary searching to find which are in and which are out.

It follows from the proof of Theorem 9 that any associatively selective set for which there is an infinite set of lengths at which we know it is not empty is in fact non-coNP-immune. This is a *very* weak type of partial census information.

Definition 4. *We say a set B is hintable if there is a tally set, $T \subseteq 1^*$, such that $T \in \mathrm{P}$ and $(\forall i)[1^i \in T \implies \|B^{=i}\| \neq 0]$.*

Theorem 10. *Any infinite, hintable* A*-selective (or even* A_ℓ*-selective) set has an infinite* coNP *subset (i.e.,* A_ℓ-sel $\cap$ Hintable *is not* coNP*-immune).*

5 Are All P-Selective Sets Associatively P-Selective?

Many of our results give simplicity properties of A-sel. It is natural to wonder whether in fact A-sel is all of P-sel. We will prove, as Corollary 3, that if P = NP, then these classes coincide.

Theorem 11. *Every* P*-selective set has a selector function in* $\mathrm{FP}^{\mathrm{NP}}$ *that is commutative and associative.*

Proof: Let A be a P-selective set. Let f be a P-selector of A, without loss of generality assume f to be commutative. For every pair of strings x and y, ω is called a connector of x and y if and only if either (a) $f(x,\omega) = \omega$ and $f(\omega, y) = y$ or (b) $f(x,\omega) = x$ and $f(\omega, y) = \omega$. Note that for any two strings x and y, there

always exist connectors of x and y, namely x and y. Define the function g by setting, for all x and y,

$$g(x,y) = \begin{cases} y & \text{if the lexicographically smallest connector } \omega \text{ of } x \text{ and } y \\ & \text{yields } f(x,\omega) = \omega \text{ and } f(\omega,y) = y, \\ x & \text{otherwise.} \end{cases}$$

It is not hard to see that g is an $\mathrm{FP}^{\mathrm{NP}}$ function. We will now argue that g is a commutative and associative selector for A. By definition, g is clearly commutative.

We will show that g is a selector for A by showing that for any two strings x and y, $g(x,y)$ satisfies the conditions (1) $g(x,y) = x \vee g(x,y) = y$ and (2) $(x \in A \vee y \in A) \implies g(x,y) \in A$ (see Definition 1). Let x and y be any two strings. Condition (1) holds by definition of g. If $x = y$ or $f(x,y) = g(x,y)$ then also (2) is trivially satisfied. So suppose $x \neq y$ and $f(x,y) \neq g(x,y)$. Without loss of generality assume $f(x,y) = x$ and $g(x,y) = y$. Hence the (lexicographically) smallest connector of x and y, call it ω, yields $f(x,\omega) = \omega$ and $f(\omega,y) = y$. Since $f(x,y) = x$, $f(x,\omega) = \omega$, and $f(\omega,y) = y$ it follows from the fact that f is a selector of A that either $x,y,\omega \in A$ or $x,y,\omega \notin A$. Hence (2), i.e., $(x \in A \vee y \in A) \implies g(x,y) \in A$, holds.

It remains to show that g is an associative function. Assume that g is not associative. Hence there exist three disjoint strings x, y, and z such that $g(x,g(y,z)) \neq g(g(x,y),z)$. (Note that if $x = y$ or $y = z$ or $x = z$ then $g(x,g(y,z)) = g(g(x,y),z)$.) It is not hard to see that $g(x,g(y,z)) \neq g(g(x,y),z)$ is equivalent to

$$\begin{aligned}(g(x,y) = y \wedge g(y,z) = z \wedge g(x,z) = x) \vee \\ (g(x,y) = x \wedge g(y,z) = y \wedge g(x,z) = z) \qquad (*).\end{aligned}$$

Informally put, $g(x,g(y,z)) \neq g(g(x,y),z)$ is equivalent to the statement that x, y, and z induce a directed cycle of length three in the (infinite) tournament induced by g. Let ω be the lexicographically smallest connector among all connectors of x and y, among all connectors of y and z, and among all connectors of x and z. Let $a,b \in \{x,y,z\}$, $a \neq b$, be such that ω is a smallest connector of a and b. Without loss of generality assume $g(a,b) = b$. Hence $f(a,\omega) = \omega$ and $f(\omega,b) = b$. Let c be the string in $\{x,y,z\} \setminus \{a,b\}$. If $f(c,\omega) = c$ then ω is also the (lexicographically) smallest connector of a and c and thus $g(a,c) = c$, contradicting $(*)$. If $f(c,\omega) = \omega$ then ω is also the (lexicographically) smallest connector of b and c and thus $g(b,c) = b$, contradicting $(*)$. This shows that g is associative and completes our proof. ❑

Corollary 3. $\mathrm{P} = \mathrm{NP} \implies \text{P-sel} = \text{A-sel}$.

Theorem 6 provides a sufficient condition, based on an algebraic property for selector functions, for P-sel $\subseteq \mathrm{P}/\mathcal{O}(n)$. If one were interested only in *structural complexity-class-collapse* sufficient conditions, Theorem 6 would be no improvement over the $\mathrm{P} = \mathrm{NP}$ sufficient condition implicit in the result of Hemaspaandra

and Torenvliet that P-sel $\subseteq$ NP/$\mathcal{O}(n)$ (and thus P-sel $\subseteq$ P/$\mathcal{O}(n)$ if P = NP), as in light of Corollary 3 the best known structural complexity-class-collapse condition sufficient to imply P-sel = A-sel is also the collapse P = NP. However, we feel that this is the wrong view, and that "P-sel = A-sel" is probably a fundamentally different type of assumption than P = NP. For example, if P-sel = A-sel were in fact equivalent to P = NP, then P = NP would (trivially) be not just a sufficient condition for P-sel = A-sel but also would be a necessary condition. In fact, not only is P = NP not known to be necessary for P-sel = A-sel, but in fact no structural complexity-class-collapse condition—not even very weak collapses like P = UP or P = ZPP—is known to be necessary. Our point here is that, though by Corollary 3 P = NP is one way to achieve P-sel = A-sel, we conjecture that it is unlikely to characterize P-sel = A-sel. And thus, our P-sel = A-sel sufficient condition for P-sel $\subseteq$ P/$\mathcal{O}(n)$ is best viewed as a new algebraic sufficient condition quite different from the known (and extremely demanding) structural complexity-class-collapse sufficient conditions.

Finally, we mention two open issues. First, can one prove a complete or partial converse of Corollary 3? The second issue is the following. One would ultimately like to know whether all P-selective sets have linear deterministic advice, i.e., whether P-sel $\subseteq$ P/$\mathcal{O}(n)$. This paper gave a new sufficient condition for that, namely, P-sel $\subseteq$ P/$\mathcal{O}(n)$ if all P-selective sets have associative (or even merely length-associative) selector functions. On the other hand, the task of constructing an oracle relative to which P-sel $\subseteq$ P/$\mathcal{O}(n)$ fails has resisted any progress for half a decade. This paper suggests a weaker question that can be tackled first: trying to construct an oracle relative to which some P-selective sets have no length-associative selector functions.

Acknowledgments

We are deeply grateful to Klaus Wagner for valuable suggestions. We thank Dieter Kratsch, Haiko Müller, Till Tantau, and the anonymous referees for their helpful comments.

References

1. E. Allender and L. Hemachandra. Lower bounds for the low hierarchy. *Journal of the ACM*, 39(1):234–251, 1992.
2. J. Balcázar and U. Schöning. Logarithmic advice classes. *Theoretical Computer Science*, 99(2):279–290, 1992.
3. J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoretical Computer Science*, 34:17–32, 1984.
4. E. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *Journal of Computer and System Sciences*, 53(2):194–209, 1996.
5. L. Hemaspaandra, H. Hempel, and A. Nickelsen. Algebraic properties for deterministic and nondeterministic selectivity. In Preparation.
6. L. Hemaspaandra, H. Hempel, and A. Nickelsen. Algebraic properties for deterministic selectivity. Technical Report TR-746, Department of Computer Science, University of Rochester, Rochester, NY, April 2001.

7. L. Hemaspaandra, A. Hoene, A. Naik, M. Ogiwara, A. Selman, T. Thierauf, and J. Wang. Nondeterministically selective sets. *International Journal of Foundations of Computer Science*, 6(4):403–416, 1995.
8. L. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25(4):697–708, 1996.
9. L. Hemaspaandra, M. Ogihara, and G. Wechsung. Reducing the number of solutions of NP functions. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 394–404. Springer-Verlag *Lecture Notes in Computer Science #1893*, August/September 2000.
10. L. Hemaspaandra and L. Torenvliet. Optimal advice. *Theoretical Computer Science*, 154(2):367–377, 1996.
11. L. Hemaspaandra, M. Zaki, and M. Zimand. Polynomial-time semi-rankable sets. In *Journal of Computing and Information*, 2(1), Special Issue: *Proceedings of the 8th International Conference on Computing and Information*, pages 50–67, 1996. CD-ROM ISSN 1201-8511/V2/#1.
12. R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 302–309. ACM Press, April 1980. An extended version has also appeared as: Turing machines that take advice, *L'Enseignement Mathématique*, 2nd series, 28, 1982, pages 191–209.
13. K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26:209–221, 1983.
14. K. Ko. On helping by robust oracle machines. *Theoretical Computer Science*, 52:15–36, 1987.
15. K. Ko and U. Schöning. On circuit-size complexity and the low hierarchy in NP. *SIAM Journal on Computing*, 14(1):41–51, 1985.
16. J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM Journal on Computing*, 28(1):311–324, 1998.
17. J. Moon. *Topics on Tournaments*. Holt, Rinehart, and Winston, 1968.
18. A. Naik, J. Rogers, J. Royer, and A. Selman. A hierarchy based on output multiplicity. *Theoretical Computer Science*, 207(1):131–157, 1998.
19. A. Naik and A. Selman. Adaptive versus nonadaptive queries to NP and P-selective sets. *Computational Complexity*, 8(2):169–187, 1999.
20. A. Nickelsen. *Polynomial-Time Partial Information Classes*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 2000.
21. M. Ogihara. Functions computable with limited access to NP. *Information Processing Letters*, 58:35–38, 1996.
22. U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27:14–28, 1983.
23. A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.
24. A. Selman. Some observations on NP real numbers and P-selective sets. *Journal of Computer and System Sciences*, 23:326–332, 1981.
25. A. Selman. Analogues of semirecursive sets and effective reducibilities to the study of NP complexity. *Information and Control*, 52:36–51, 1982.
26. A. Selman. Reductions on NP and P-selective sets. *Theoretical Computer Science*, 19(3):287–304, 1982.
27. Till Tantau, March 2001. Personal Communication.
28. S. Toda. On polynomial-time truth-table reducibilities of intractable sets to P-selective sets. *Mathematical Systems Theory*, 24:69–82, 1991.

Parallelizability of Some P-Complete Geometric Problems in the EREW-PRAM*

Carla Denise Castanho[1], Wei Chen[2], Koichi Wada[1], and Akihiro Fujiwara[3]

[1] Nagoya Institute of Technology, Showa, Nagoya 466-8555, Japan
[2] Nanzan University, Seirei-cho 27, Seto-shi, Aichi-ken 489-0863, Japan
[3] Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan
caca@elcom.nitech.ac.jp, chen@it.nanzan-u.ac.jp
wada@elcom.nitech.ac.jp, fujiwara@cse.kyutech.ac.jp

Abstract. P-complete problems seem to have no parallel algorithm which runs in polylogarithmic time using a polynomial number of processors. A P-complete problem is in class EP (Efficient and Polynomially fast) if and only if there exists a cost optimal algorithm to solve it in $T(n) = O(t(n)^{\epsilon})$ $(\epsilon < 1)$ using $P(n)$ processors such that $T(n) \times P(n) = O(t(n))$, where $t(n)$ is the time complexity of the fastest sequential algorithm which solves the problem. The goal of our research is to find EP parallel algorithms for P-complete problems. In this paper we consider two P-complete geometric problems in the plane. First we consider the convex layers problem of a set S of n points. Let k be the number of the convex layers of S. When $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 < \epsilon < 1)$ we can find the convex layers of S in $O(\frac{n \log n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$. Next, we consider the envelope layers problem of a set S of n line segments. Let k be the number of the envelope layers of S. When $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 < \epsilon < 1)$, we propose an algorithm for computing the envelope layers of S in $O(\frac{n\alpha(n)\log^3 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$, and $\alpha(n)$ is the functional inverse of Ackermann's function which grows extremely slowly. The computational model we use in this paper is the $EREW\text{-}PRAM$. Our first algorithm, for the convex layers problem, belongs to EP, and the second one, for the envelope layers problem, belongs to the class EP if a small factor of $\log n$ is ignored.

1 Introduction

In parallel computational theory, one of the primary measures of parallel complexity is the class NC. A problem is said to be in NC if there exists an algorithm which solves the problem in polylogarithmic time using polynominal number of

* This work was partly supported by the Hori Information Promotion Foundation (1999), a Scientific Research Grant-In-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan, under grant No. 10205209, and a Scientific Research Grant-in-Aid for Encouragement of Young Scientists from the Ministry of Education, Culture, Sports, Science and Technology, under grant No. 12780236.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 59–63, 2001.
© Springer-Verlag Berlin Heidelberg 2001

processors. The class of P-complete problems consists of the most likely candidates of P that are not in NC. However, polylogarithmic time complexity is not so important when considering practical parallel computation. Actually, the number of processors available is usually small in comparison with the size of a problem. Thus, in practice cost optimality turns to be the most important measure for parallel algorithms. The cost of a parallel algorithm is defined as the product of the running time and the number of processors required by the algorithm. A parallel algorithm is called cost optimal if its cost is of the same order as the time complexity of the fastest known sequential algorithm for the same problem. Kruskal et al. [8] proposed class EP (Efficient and Polynomially fast). A problem is in EP if and only if there exists a cost optimal algorithm to solve it in $T(n) = O(t(n)^{\epsilon})$ $(\epsilon < 1)$ using $P(n)$ processors such that $T(n) \times P(n) = O(t(n))$, where t(n) is the time complexity of the fastest sequential algorithm which solves the problem.

In this paper, we consider EP algorithms for two P-complete geometric problems, the convex layers problem and the envelope layers problem on EREW-PRAM. Given a set S of n points in the plane, S can be divided into a sequence $CL(S)$ of convex hulls: $CL_1(S), CL_2(S), \ldots, CL_k(S)$, where $CL_i(S)$ $(1 \leq i \leq k)$ is the convex hull of $\cup_{j=i+1}^{k} CL_j(S)$. The envelope layers of a set S of (opaque) n line segments are analogous to the convex layers of a set of points, with convex hulls replaced by upper envelopes. The upper envelope of S is the collection of segment portions visible from the point $(0, +\infty)$, which consists of $O(n\alpha(n))$ subsegments of S. These two problems are P-complete [4,6]. They are still P-complete even if $1 \leq k \leq n^{\frac{1}{2}}$.

Chazelle [1] proposed an optimal sequential algorithm for the convex layers problem which runs in $O(n \log n)$ time. In [5] Fujiwara et al. considered the convex layers problem under a very strong constraint, in which they proved that if all points of S lie on d horizontal lines, when $d \leq n^{\epsilon}$ $(0 < \epsilon \leq \frac{1}{2})$ the problem is still P-complete. They proposed an EP algorithm for the problem which runs in $O(\frac{n \log n}{p})$ time using p processors if $d \leq n^{\epsilon}$ $(0 < \epsilon \leq \frac{1}{2})$ and $1 \leq p \leq n^{\epsilon}$ in the $EREW\text{-}PRAM$. That is, to achieve cost optimality, there must be d horizontal lines such that each line must pass through more than $n^{\frac{1}{2}}$ points in average, what is unlikely in most cases. Besides, the parameter d does not represent the substantial complexity of the problem. In this paper we present a new EP parallel algorithm for computing the convex layers of a set S of n points. Let k be the number of the convex layers of S. When $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 < \epsilon < 1)$ our algorithm runs in $O(\frac{n \log n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$, in the $EREW\text{-}PRAM$, and it is cost optimal.

Let S be a set of n (opaque) line segments in the plane. Hershberger [6] gave an $O(n\alpha(n) \log^2 n)$ algorithm for computing the envelope layers of S, where $\alpha(n)$ is the functional inverse of Ackermann's function which grows extremely slowly. Here, we also give an algorithm for the envelope layers problem. Let k be the number of envelope layers of S. When $0 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 < \epsilon < 1)$ our algorithm runs in $O(\frac{n\alpha(n) \log^3 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$, in the $EREW\text{-}PRAM$. If we ignore a factor of $\log n$ our algorithm belongs to the class

EP. In the following sections, we give the outline of our algorithms. The details can be found in [2].

2 Finding Convex Layers of Points

In our algorithm the following operations are required: (I) storing a convex layer, (II) searching for an edge of a convex layer, (III) finding the common tangents of two convex layers, (IV) deleting a set of contiguous edges from a convex layer, (V) given two convex polygons P and Q finding the convex hull of P and Q, and (VI) given x convex polygons, P_1, P_2, ..., P_x with f vertices each, finding the common tangents of P_i and P_j for all pairs of i, j $(1 \leq i < j \leq x)$. Let the sizes of the convex layers and convex polygons are $O(n)$. We keep convex layers in balanced tree. Operation (I) can be done in $O(\log n)$ using n processors [7], operation (II) and (III) can be done in $O(\log n)$ time sequentially [9]. Operations (IV) and (V) can be done in $O(\log n)$ time sequentially [2]. Operation (VI) can be done in $O(\frac{\log x \log^2 f}{\log^2 p - \log^2 x^2})$ $(p \leq fx^2)$ time using p processors [3]. Let S be a set of n points in the plane and k be the number of the convex layers of S. We sort S by its x-coordinates. The following algorithm finds the convex layers of S.

Algorithm $ComputeCL(S)$
[Input] A set $S = (p_1, p_2, \ldots, p_n)$ of n points in the Euclidean plane sorted by their x-coordinates.
[Output] A set $CL(S) = (CL_1(S), CL_2(S), \ldots, CL_k(S))$ $(1 \leq k \leq n)$ of the convex layers of S, where $CL_i(S)$ $(1 \leq i \leq k)$ is the ith convex layer of S.
(Phase 1) Divide S into $S_1, S_2, \ldots, S_{n^{\frac{1-\epsilon}{2}}}$ subsets such that S_i $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ contains $n^{\frac{1+\epsilon}{2}}$ points and the x-coordinate of any point of S_i is less than the x-coordinate of any point of S_{i+1}.
(Phase 2) In parallel, for each subset S_i $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ compute the convex layers of S_i, denoted as $CL(S_i) = (CL_1(S_i), CL_2(S_i), \ldots, CL_{k_i}(S_i))$ by Chazelle's sequential algorithm, where k_i is the number of layers in S_i. Store each $CL_j(S_i)$ $(1 \leq j \leq k_i)$ $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ in balanced tree.
(Phase 3) Let $S'_i = S_i$ $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$. Let $x = 1$. Find $CL_x(S)$, that is, the xth convex layer of S, repeatedly as follows.

(a) Find the convex hull of the outermost layers of $S'_1, S'_2, \ldots, S'_{n^{\frac{1-\epsilon}{2}}}$, which obviously is $CL_x(S)$.

(b) Revise S'_i $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ as follows: for each i $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ in parallel, delete the vertices of $CL_x(S)$ from S'_i, and reconstruct the convex layers of S'_i.

(c) Set $x = x + 1$. While all of $S'_1, S'_2, \ldots, S'_{n^{\frac{1-\epsilon}{2}}}$ are not completely empty return to (a). ∎

In the above algorithm, Phase 1 can be executed in $O(1)$ time using n processors, if S is stored in an array. In Phase 2, $CL(S_i)$ can be computed by Chazelle's sequential algorithm [1] in $O(n^{\frac{1+\epsilon}{2}} \log n)$ time. Therefore, all the convex layers of $S_1, S_2, \ldots, S_{n^{\frac{1-\epsilon}{2}}}$ can be computed in $O(n^{\frac{1+\epsilon}{2}} \log n)$ time using $n^{\frac{1-\epsilon}{2}}$ processors.

We can store $CL_j(S_i)$ $(1 \leq j \leq k_i,\ 1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ in balanced tree in $O(\log |S_i|)$ time using $|S_i|$ processors. Now let us consider Phase 3. Phase 3(a) can be done in $O(\log n)$ time using $O(n^{1-\epsilon})$ processors[3]. Using the operations we defined, Phase 3(b) can be done in $O(k \log n)$ time using $O(n^{\frac{1-\epsilon}{2}})$ processors. Since the instructions of Phase 3 are repeated k times, where k is the number of convex layers of S, Phase 3 takes totally $O(k^2 \log n)$ time using $O(n^{\frac{1-\epsilon}{2}})$ processors. Thus, the whole algorithm $ComputeCL(S)$ can be executed in $O(max(n^{\frac{1+\epsilon}{2}} \log n, k^2 \log n))$ time using $n^{\frac{1-\epsilon}{2}}$ processors. Thus by using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$, it takes $O(max(\frac{n \log n}{p}, \frac{n^{\frac{1-\epsilon}{2}} k^2 \log n}{p}))$ time in the $EREW\text{-}PRAM$. Therefore, when $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 < \epsilon < 1)$, the algorithm $ComputeCL(S)$ is cost optimal and runs in $O(\frac{n \log n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$.

3 Finding Envelope Layers of Line Segments

Let S be a set of n (opaque) line segments, and k be the number of the envelope layers of S. From the duality of lines and points, the convex layers of lines can be found by the algorithm for the convex layers of points. The main idea of our algorithm here is to reduce the problem of line segments into that of lines by using a segment tree.

The plane can be partitioned into $2n + 1$ slabs by drawing vertical lines through all the segments endpoints of S. The segment tree of S, denoted as $ST(S)$, is built as follows [10]. Construct a complete binary tree with $2n$ leaves. Then let each leaf of $ST(S)$ represent one slab taken in left-to-right order, and each internal node represent the union of its descendants' slabs. Whether original slabs or a union of them, are called *canonical slabs*. Descending from the root to the leaves, associate with each node $v \in ST(S)$ a set $S[v]$, where $S[v]$ consists of the segments or subsegments of S that have its endpoints on the boundary of the canonical slab represented by node v, which have not been associated with any of v's ancestors in $ST(S)$. For any node v, the line segments in $S[v]$ have the same left and right endpoints, respectively, therefore, they can be considered as lines. On the other hand, for any two nodes x and y on the same level of $ST(S)$, the subsets associated with nodes x and y are separated. Thus, for the sets associated on the same level their envelopes can be found in parallel by the algorithm for the convex layers of points.

Algorithm $ComputeEL(S)$
[Input] A set S of n (opaque) line segments in the plane.
[Output] A set $EL(S) = (EL_1(S), EL_2(S), \ldots, EL_k(S))$ $(1 \leq k \leq n)$ of the envelope layers of S, where k is the number of layers, and $EL_i(S)$ $(1 \leq i \leq k)$ is the ith envelope layer of S.
(Phase 1) Use a segment tree, denoted as $ST(S)$, which has $m = O(\log n)$ levels, to cut the n segments of S into $O(n \log n)$ subsegments and divide them into m separate groups, say $G_1, G_2, \ldots, G_m$, where group G_i $(1 \leq i \leq m)$ corresponds to the subsegments associated on level i of $ST(S)$. Reduce the envelope layers problem of G_i to the convex layers problem of $|G_i|$ points.

(Phase 2) For each i $(1 \leq i \leq m)$ in parallel, find $EL(G_i)$ the envelope layers of G_i by using our algorithm for the convex layers.
(Phase 3) Find the envelope of $EL(G_1), \ldots, EL(G_m)$ as follows.

(1) Cutting $EL(G_1), \ldots, EL(G_m)$, by $n^{\frac{1-\epsilon}{2}}$ vertical lines into $n^{\frac{1-\epsilon}{2}}$ separate groups $H_1, H_2, \ldots, H_{n^{\frac{1-\epsilon}{2}}}$. Then, for each i $(1 \leq i \leq n^{\frac{1-\epsilon}{2}})$ in parallel, compute the envelope layers of H_i, $EL(H_i) = (EL_1(H_i), EL_2(H_i), \ldots, EL_{h_i}(H_i))$, where h_i is the number of layers in H_i, by Hershberger's sequential algorithm [6].

(2) Let $k = max(h_1, h_2, \ldots, h_{n^{\frac{1-\epsilon}{2}}})$. Obtain $EL_t(S)$ $(1 \leq t \leq k)$, i.e. the tth envelope layer of S, by concatenating $EL_t(H_1)$, $EL_t(H_2)$, $\ldots$, $EL_t(H_{n^{\frac{1-\epsilon}{2}}})$ ($EL_t(H_i)$ is empty if $t > h_i$). ∎

Phase 1 can be done in $O(\log n)$ time using n processors. When $1 \leq k \leq n^{\frac{\epsilon}{2}}$ $(0 < \epsilon < 1)$, Phase 2 can be done in $O(\frac{n \log^2 n}{p})$ time using p processors, and Phase 3 can be done in $O(\frac{n\alpha(n) \log^3 n}{p})$ time using p processors, where $1 \leq p \leq n^{\frac{1-\epsilon}{2}}$. The proof can be found in [2].

References

1. Chazelle, B.: On the convex layers of a planar set. IEEE Transactions on Information Theory. **vol.31(4)** (1995) 509-517.
2. Castanho,C. D., Chen,W., Wada,K.: Polynomially fast parallel algorithms for some P-complete problems. Technical Report, WadaLab-00-9, Dept. of Electrical and Computer Engineering, Nagoya Institute of Technology, Sep. (2000).
3. Chen,W., Nakano,K., Masuzawa,T., Tokura,N.: Optimal parallel algorithms for finding the convex hull of a sorted point set (in Japanese). Trans. IEICE. **J75-D-I, 9** (1992) 809-820.
4. Dessmark,A., Lingas,A., Maheshwari,A.: Multi-list ranking: complexity and applications. In 10th Annual Symposium on Theoretical Aspects of Computer Science. LNC **vol.665** (1993) 306-316.
5. Fujiwara,A., Inoue,M., Masuzawa,T.: Practical parallelizability of some P-complete problems. Technical Report of IPSF. **vol.99 (72)** (1999) 9-16.
6. Hershberger,J.: Upper envelope onion peeling. Computational Geometry: Theory and Applications. **vol.2** (1992) 93-110.
7. JáJá,J.: An Introduction to Parallel Algorithms. Addison-Wesley (1992).
8. Kruskal,C. P., Rudolph,L., Snir,M.: A complexity theory of efficient parallel algorithms. Theoretical Computer Science. **vol.71** (1990) 95-132.
9. Overmars,M. H., Leeuwen,J. V.: Maintenance of configurations in the plane. J. Comput. System Sci **vol.23** (1981) 166-204.
10. Preparata,F. P., Shamos,M. L.: Computational Geometry: An Introduction. Springer-Verlag (1985).

Enhanced Sequence Reconstruction with DNA Microarray Application*

Extended Abstract

Samuel A. Heath and Franco P. Preparata**

Computer Science Department, Brown University

Abstract. DNA sequencing by hybridization is potentially a powerful alternative to standard gel electrophoresis techniques. An important aspect of the approach is the design of the probing scheme and of the associated sequence reconstruction algorithm. Recently a novel probing scheme, whose performance is within a constant factor of the information theory bound, has settled the issue of asymptotic optimality. Thus, the research focus has shifted to the fine tuning of actual performance, with enormous potential for the life sciences. In this paper we discuss a new algorithmic device, called voting upon failure, which, exploiting the knowledge acquired in the course of the sequence reconstruction process, achieves typically a 20% performance improvement over the previous best technique, and comes at 90%-confidence within a factor 0.5 of the information-theory bound.

1 Introduction

A central application in molecular biology is the sequencing of DNA, i.e., the determination of the sequence of nucleotides of a chosen (fragment of a) DNA molecule. In recent years, a radically new technique has been proposed as an alternative to the traditional sequencing by gel electrophoresis. This technique, called *Sequencing by Hybridization* (SBH), proposed independently by several different research teams [BS91, D89, L88, P89, P91] is based on the use of a chip fabricated on a glass substrate. The active area of the chip is structured as a matrix, in each region of which (a *feature*) numerous copies of a specific oligonucleotide (a short sequence of DNA nucleotides) are implanted. The chip is immersed under controlled conditions within a solution of suitably labeled target DNA sequence. A copy of the target DNA will bind (hybridize) to an oligonucleotide if the oligonucleotide is complementary, in the Watson-Crick sense, to one of its subsequences. The labeling of the target allows visualization of the chip features containing binding oligonucleotides, thereby yielding a method for automatically probing the target sequence for specific subsequences.

* This work was partially supported by the National Science Foundation under Grant DBI-9983081

** Computer Science Department, Brown University, 115 Waterman Street, Providence, RI 02912-1910, USA.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 64–74, 2001.
© Springer-Verlag Berlin Heidelberg 2001

In summary, Sequencing by Hybridization consists of two fundamental steps. The first, biochemical in nature, is the acquisition, by complementary hybridization with a complete library of probes, of all subsequences (of a selected pattern) of a given unknown target sequence; the set of such subsequences is called the sequence *spectrum*. The second step, combinatorial in nature, is the algorithmic reconstruction of the sequence from its spectrum. Most traditional schemes were restricted to probes in the form of strings of k symbols (k-mers), which could achieve no better performance (expressed as the length of reliably reconstructible sequences) than the square root of the information-theoretic bound [DFS94, PFU99], as prominently noted in a review paper by well-known biologist Ed Southern [S96].

Recently a novel approach has been proposed [PFU99],[PU00], whose performance is within a small factor of the optimum. The central feature of this approach is the adoption of a probing pattern where natural nucleotides are separated by gaps of "wild-cards", intended to hybridize non-specifically with any natural base and proposed to be realized with artificial universal bases [LB94], i.e., bases that stack without binding.

It must be stressed that, since the question of the asymptotic performance has been settled, the focus of research in SBH sequence reconstruction has shifted to the refinement of the algorithmic procedures, in order to fine-tune computational methods which have enormous applicational potential in the life sciences. In this paper we shall briefly review the combinatorics underlying the approach, the known algorithmic technique, and illustrate a new property which enables a significant enhancement of the achievable performance.

2 Review of the Gapped-Probe Approach

A key feature of SBH is the adopted probing pattern, i.e., the structure of the spectrum probes. Notationally, a probing pattern is specified by a binary string (beginning and ending with a 1) where a 1 denotes a natural base and a 0 a universal base. For $s + r = k$, an (s, r) direct probing scheme has the pattern $1^s(0^{s-1}1)^r$, while an (s, r) reverse scheme has the pattern $(10^{s-1})^r 1^s$. For notational convenience a probe is viewed as a string of length $(r + 1)s$ over the extended alphabet $\mathcal{A} =$ { A,C,G,T,*}, where $*$ denotes the "wild-card".

In association with any such probing scheme the reconstruction of the sequence is algorithmically carried out symbol-by-symbol from one end to the other [PFU99, PU00]. The algorithmic primitive is the interrogation of the spectrum with a query of the form $u*$, where u is a string of length $(r+1)s-1$ over $\mathcal{A}$, which returns all matching probes. Performance is naturally measured as the length of sequences that are reconstructible with a given confidence level, under the standard hypothesis that the target sequence is generated by a maximum-entropy memoryless source (i.i.d. symbols).

The algorithm interrogates the spectrum with $q*$, where q is the current $((r+1)s-1)$-symbol suffix of the reconstructed sequence. If only one probe is returned, the sequence is trivially extended; otherwise there is a potential ambiguity and

the algorithm attempts the extension of all potential paths up to a predetermined length H. The rationale is that the correct path is deterministically extended, whereas the extension of the spurious paths rests on the improbable presence of (fooling) probes in the spectrum. At the conclusion of this process, the sequence is extended with the common prefix of all surviving paths; otherwise, when such a prefix is empty, the existence of alternative paths of length H is taken as evidence that their extension would continue with no definite bound, and reconstruction fails [PU00]. The performance of such an algorithm at 90%-confidence comes within a factor slightly smaller than 0.5 of the theoretical optimum.

3 Analysis of the Failure Modes of the Reconstruction Algorithm

The best known algorithm for sequence reconstruction [PU00] has two failure modes, distinguished on the basis of the paths issuing beyond the ambiguous branching embodying the reconstruction failure. The failure modes can be classified as follows:

1. Failure Mode 1. There are two paths identical except for their initial symbol (corresponding to the branching). This failure has been caused by k fooling probes scattered, with possible overlaps, along the target sequence. On the basis of the spectrum alone there is no way to decide which of the multiple symbols detected at the branching corresponds to the correct extension.
2. Failure Mode 2. There are no two paths as specified for Failure Mode 1. If H is large enough to make probabilistic extension of a spurious path to length H extremely unlikely, then there must be an $((r+1)s-1)$-segment of a spurious path, including or following the branching position which is identical to an actual segment occurring in the sequence. Obviously, in such a case extension of this path is deterministically assured and the reconstruction fails. (Such substrings are referred to as *self-sustaining*.) The self-sustaining segment agrees, entirely or partially, with an equally positioned segment of the correct path, the disagreements being concealed by fooling strings also occurring in the sequence.

Failure Mode 1 is well understood and discussed elsewhere [PU00]. Suffice it here to recall that, denoting with $\alpha = 1 - e^{-m/4^k}$ the probability of occurrence of a specific probe in a target sequence of length m, an accurate estimate of the probability of failure by Mode 1 has the expression:

$$P_1 = 1 - e^{-3m\alpha^k \left(1+\frac{4^{r+1}}{3m}\right)^r \left(1+\frac{4^s}{3m}\right)^{s-1}}$$

Before analyzing Failure Mode 2, it is appropriate the study the two following examples (observed in simulations of random target sequences for the probing pattern (4,4): 1111 0001 0001 0001 0001). We have:

Example 1

```
GCAT TACT GAAC GGTT ACA [T] TGAT  ATGA  TGC...
                        [G] TCAC  AAGC  TAT...

... AC g a a c g g t t a c a g  t c a c  a a g C TAT ...
                           *   *  *    *
```

Example 2

```
ATAT GCAC GCCT [G] AGT GTAT ATTC TGGC  CACT   GCCA CG ...
               [T] AGT GTAT ATCA TAGG  TACG   TGAC AG ...

...  TCGC GCGG T AAC g t a t a t c a t a g g  t a c g  t g a C AG ...
             *               * *  *  *  *   *  * * *
```

In the above examples the first two lines illustrate the branching event (symbols within brackets "[ ]"). Adequate prefixes of the two paths issuing from the branching are used to locate matching segments in the original sequence. The top path agrees with the putative sequence, whereas only a suffix of the bottom path agrees with an actual substring of the sequence. The latter is shown in the third line: the 19-symbol subsegment shown in lower case letters (referred to as the *self-sustaining segment*) supports the indefinite extension of the spurious path. This segment disagrees with the top (correct) segment in the positions labelled "$*$", and these disagreements must be concealed by fooling probes. Note that the self-sustaining segment may not begin more than $19 = (r+1)s - 1$ positions to the left of the branching; in Example 1 it starts on the left of the branching, in Example 2 on its right.

Next we analyze in detail this failure mode. The self-sustaining segment has length $\nu = (r+1)s - 1$. This length is indeed sufficient for indefinite path extension. If the branching position is conventionally denoted as 0, the first position J of the self-sustaining segment, referred to as the *offset*, must satisfy the condition $J \geq -\nu$. The failure events corresponding to each value J of the offset can be constructed by first selecting two positions in the sequence (in $\binom{m}{2} \approx m^2/2$ ways), so that its probability, for some coefficient π_J, can be expressed as

$$\frac{m^2}{2}\pi_J$$

The term π_J, referred to as the *multiplier*, depends upon the disagreements between the self-sustaining segment and the correct path.

The analysis of π_J is greatly aided by an s-row arrays (shown in Figure 1 for $s = 5$ and $r = 3$), whose cells correspond to sequence positions with reference to the branching position and are numbered starting from $-\nu = -(r+1)s + 1$ in column-major order. In the array we identify as the *critical set* the set of cells $0, s, 2s, \ldots, rs, rs+1, rs+2, \ldots, (r+1)s - 1$ (shown crossed-out in Figure 1),

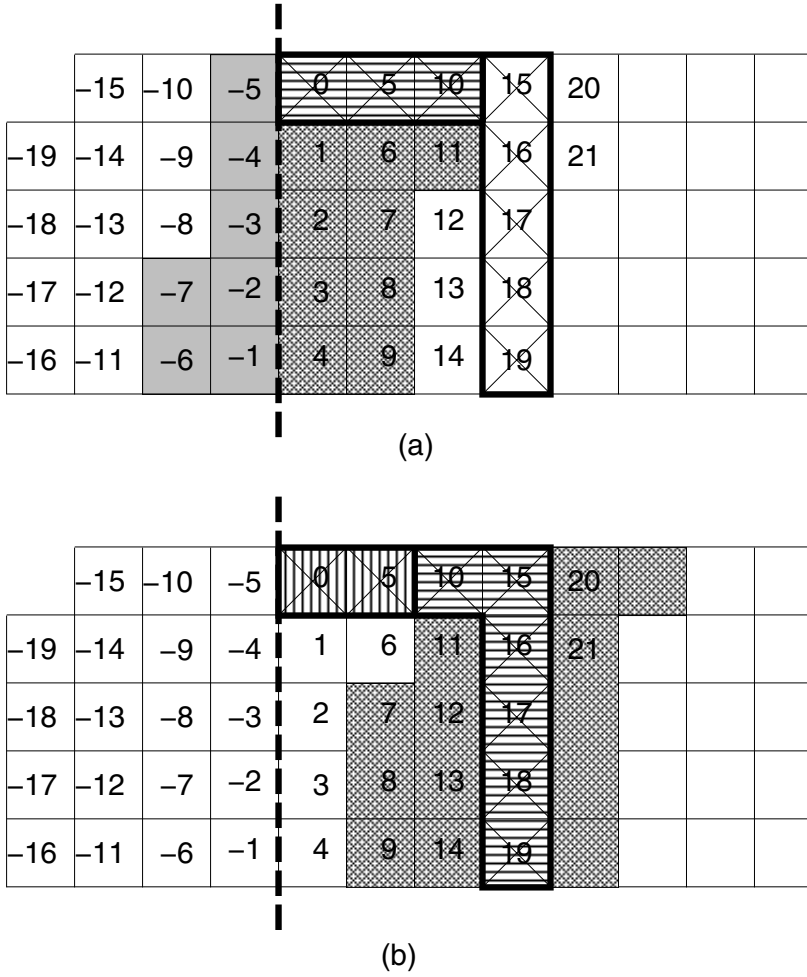


Fig. 1. Illustration of offset domains fot $J = -7$ and $J = 7$. Evidence is given to constrained symbols (light shading), critical set (crossed), and selectable rows (heavier shading)

which represent the positions of the last symbols of the fooling probes necessary to support the branching. A value J of offset identifies in this array the domain (offset domain) of the self-sustaining segment (two typical offset domains are shown in Figure 1 for $J = -7$ and $J = 7$). For each value of the offset the following parameters are significant:

1. The number of deterministically constrained sequence symbols at the self-sustaining segment site. Since no fooling probe can conceal disagreements that precede the branching, all the symbols preceding the branching are constrained (cells shown lightly shaded in Figure 1(a)).

2. The size of the intersection with the critical set. Each cell of this intersection corresponds to a fooling probe which is to match the self-sustaining segment. (Such sets, shown horizontally hatched, have size 3 in 1(a) and size 6 in 1(b).)
3. The number of cells of the critical set with value smaller than J (such a set is empty for $J \leq 0$ and is shown vertically hatched in Figure 1(b)). Each such cell identifies a weak fooling probe.

All cells in the self-sustaining segment domain with positive index and not belonging to the critical set are called *selectable* and may be sites of disagreements between the correct path and the self-sustaining segment (shown heavily shaded in Figure 1). The usefulness of the above diagram derives from the fact that, due to the pattern of fooling probes required by a disagreement (refer to the critical set as an example), rows of selectable cells are independent and contribute multiplicatively to π_J.

It follows that the basic components of the analysis are the rows of selectable cells. Consider a row of u selectable cells (u-cell row). If there are no disagreements, then the row contributes $(1/4)^u$ to the multiplier; otherwise, let h ($h = 1, \ldots, u$) be the leftmost cell of the row where a disagreement occurs. In such a case the contribution of the row is $(1/4)^{h-1}(3/4)\alpha^{u-h+1}$, since $h-1$ symbols agree (each with probability $1/4$), the h-th symbol disagrees (with probability $3/4$) and must be confirmed by $u-h+1$ fooling probes, regardless of the identity of the symbols in cells to its right on the row. Summing over h we obtain

$$\frac{1}{4^u} + \sum_{h=1}^{u} \frac{1}{4^{h-1}} \frac{3}{4} \alpha^{u-h+1} = \frac{1}{4^u}(1 + 3\alpha \sum_{i=0}^{u-1} (4\alpha)^i)$$

We let for simplicity of notation $\beta_u = 1 + 3\alpha \sum_{i=0}^{u-1} (4\alpha)^i$, so that the contribution of a u-cell row becomes

$$\frac{\beta_u}{4^u}$$

For convenience of analysis, values of J are aggregated on the basis of the size σ of the intersection of the placement domain with the critical set. These sets, denoted Group I,...,Group VI are illustrated for our running example in Figure 2, as differently shaded domains of cells. Cells in the same domain are covered by identical analysis.

In general, these sets of offset are:

1. Group I: $J = -\nu$, $\sigma = 0$;
2. Group II: $J = -\nu + 1 + is + j$, $i = 0, 1, \ldots, r-1$, $j = 0, 1, \ldots, s-1$;
3. Group III: $J = -s + 2, -s + 3, \ldots, 0$, $\sigma = r + 1, r + 2, \ldots, k - 1$;
4. Group IV: $J = 1 + is + j$, $i = 0, 1, \ldots, r-1$, $j = 0, 1, \ldots, s-1$;
5. Group V: $J = rs + 1, rs + 2, \ldots, \nu$, $\sigma = s - 1, s - 2, \ldots, 1$;
6. Group VI: $J > \nu$,$\sigma = 0$.

In the full paper we shall report the detailed analysis of the contribution of each of these sets to the probability of failure. In this extended abstract, we illustrate in total detail the analysis of Groups I and II. It must be mentioned that, typically, the contribution of Group II accounts for about 4/5 of the total.

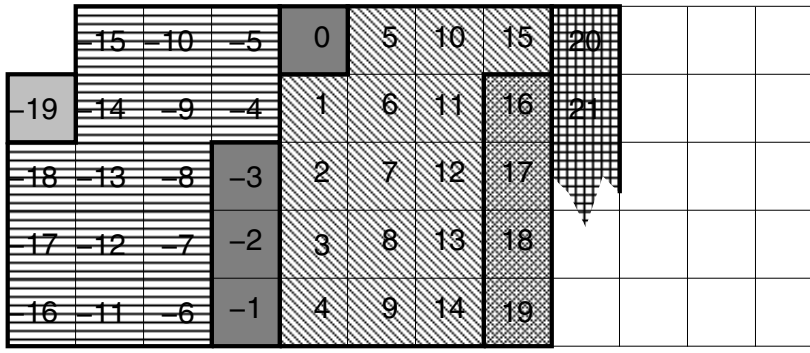


Fig. 2. Partition of values of J into Group I,. . . ,Group VI

1. Group I: $J = -\nu$, $\sigma = 0$; The self-sustaining segment fully agrees with the correct path, and the corresponding event has multiplier $\pi_{-\nu} = 3/4^{\nu+1}$, because ν symbols are fully constrained and one (the branching) can be chosen in 3 ways.
2. Group II: $J = -\nu + 1 + is + j$, $i = 0, 1, \ldots, r-1$, $j = 0, 1, \ldots, s-1$, so that $\sigma = i + 1$. We observe that the $|J|$-prefix of the segment fully agrees with the correct segment, thereby constraining $|J|$ sequence symbols. (In fact, disagreements may only occur at selectable cells with index ≥ 0.) For each value of J, we form the vector of the lengths of s selectable-cell rows. This is illustrated in Table I, whose columns, indexed by the parameter j, are the length vectors.

Table I.

	Group II				
j	1	2	3	...	s
row 1	0	0	0	...	0
row 2	i	$i+1$	$i+1$	...	$i+1$
row 3	i	i	$i+1$	...	$i+1$
...			...		
row s	i	i	i	...	$i+1$

Placement $J = -\nu + is + j$ constrains $|J|$ sequence symbols and is accompanied by $(i+1)$ fooling probes of the critical set. Also, since the multiplicative contribution of a u-cell row is $\beta_u/4^u$, we conclude that the multiplier is

$$\pi_J = \frac{3}{4}\frac{\alpha^{i+1}}{4^{-J}}\left(\frac{\beta_{i+1}}{4^{i+1}}\right)^{j-1}\left(\frac{\beta_i}{4^i}\right)^{s-j} = \frac{3\alpha(4\alpha)^i}{4^\nu}\left(\frac{\beta_{i+1}}{\beta_i}\right)^{j-1}\beta_i^{s-1}$$

If we sum over j and i we obtain the global contribution of Group II:

$$\sum_{i=0}^{r-1}\sum_{j=1}^{s}\frac{3\alpha(4\alpha)^i}{4^\nu}\left(\frac{\beta_{i+1}}{\beta_i}\right)^{j-1}\beta_i^{s-1}=\frac{1}{4^\nu}(\beta_r^s-1)$$

4 Sequence Extension Based on Voting

In the presence of the any of the previously described failures, the standard algorithm is unable to provide a reliable reconstruction. Although for Failure Mode 2 prior knowledge of the approximate sequence length may be used to discriminate between the two paths being deterministically extended (by observing that the spurious extension will terminate with a sequence of length different from the nominal one), such a criterion is inapplicable to Failure Mode 1, since both extended paths are identical beyond the branching symbol.

However, there is one important item of information that the advanced algorithm fails to use, i.e., the content, when failure occurs, of the prefix $a_{(i,\ell)}$ of the putative sequence currently constructed. In such a situation we have two alternative sets of fooling probes, respectively corresponding to the two competing extensions. We may verify which of the two sets is better represented in $a_{(i,\ell)}$ and use this information to discriminate between the two alternatives. Following intuition, since failure has occurred, one set of fooling probes (the one causing the spurious extension) is guaranteed to exist, and its members are assumed to be scattered along the sequence. Therefore we adopt the criterion to reject as spurious the path whose fooling probes are better represented in the prefix $a_{(i,\ell)}$. Failure occurs also in the case of a tie, for in such a case discrimination is not possible. We now wish to analyze probabilistically the behavior of this policy, referred to as *voting*.

Assuming that there are just two alternative extensions (the event of more simultaneous extensions is extremely rare), the discrimination is carried out on the basis of the competing sets of fooling probes. The analysis is straightforward for Failure Mode 1, since in such a case each probe set consists of the k probes sampling the branching position, because all other probes extend both paths. For Failure Mode 2, the analysis is more subtle. Referring to the preceding Example 2, each of the positions marked with an aterisk "$*$" entails a train of at most $(r+1)$ fooling probes whose last symbol occurs not beyond the end of the self-sustaining segment. The union of such trains of probes is the fooling probe set for the chosen alternative. However, the self-sustaining segment cannot be determined by the algorithm; based on the fact that long transition segments are improbable, the algorithm constructs the discriminating sets on the basis of the disagreements occurring within $(r+1)s-1$ positions from the branching. In the analysis, we shall use exactly $(r+1)s-1$ probes, which is a valid criterion since each unrequired probe detected in $a_{(i,\ell)}$ adds 1 to both probe counts.

Let $1<\ell<m$ be the position of failure and let μ_i be the common size of the competing probe sets, with $i=1,2$ corresponding to the two modes (i.e., $\mu_1=k$ and $\mu_2=(r+1)s-1$). Assume that there are j probes of the incorrect

extension in the range $[1, \ell]$. Since failure has occurred, the μ_i probes of the incorrect extension set are scattered along the sequence, and the probability that a specific probe of this set occurs in $a_{(1,\ell)}$ is $\gamma = \ell/m$. It follows that the probability of finding exactly j probes in $a_{(1,\ell)}$ is

$$p_j^{(i)}(\ell) = \binom{\mu_i}{j} \gamma^j (1-\gamma)^{\mu_i - j}$$

Let $q_j^{(i)}(\ell)$ be the probability that there are at least j probes of the correct extension set in $a_{(1,\ell)}$. Since the probability of occurrence of a specific probe is now $\delta = 1 - e^{-\ell/4^k}$, we obtain

$$q_j^{(i)}(\ell) = \sum_{h=j}^{\mu_i} \binom{\mu_i}{h} \delta^h (1-\delta)^{\mu_i - h}$$

Therefore the quantity $\sum_{j=0}^{\mu_i} p_j^{(i)}(\ell) q_j^{(i)}(\ell)$ is the probability of incorrect vote in correspondence to a failure of Mode i. We also let $\pi_i(\ell)$ denote Prob(ℓ) for Failure Mode i. We note that $\pi_1(\ell)$ is uniformly distributed in the range $[1, m]$, whereas $\pi_2(\ell)$ decreases linearly in the same range (in fact, in this case ℓ is the smaller of two values chosen in $[1, m]$). The probability that voting fails, conditional on the occurrence of the appropriate type of failure, is therefore

$$P(\text{vote failure}|\text{Mode } i) = \sum_{\ell=1}^{m} \pi_i(\ell) \sum_{j=0}^{\mu_i} p_j^{(i)}(\ell) q_j^{(i)}(\ell)$$

$$= \sum_{\ell=1}^{m} \pi_i(\ell) \sum_{j=0}^{\mu_i} \binom{\mu_i}{j} \gamma^j (1-\gamma)^{\mu - i - j} \sum_{h=j}^{\mu_i} \binom{\mu_i}{i} \delta^h (1-\delta)^{\mu_i - h} \tag{1}$$

In conclusion we obtain the probability of failure $\phi(m)$ as a function of m

$$\begin{aligned} \phi(m) &= P(\text{vote failure}|\text{Mode 1}) P_1 + P(\text{vote failure}|\text{Mode 2}) P_2 \\ &< P_1 + P_2 \end{aligned} \tag{2}$$

and the equation $\phi(m*) = \epsilon$ quantifies, for a given ϵ, the performance enhancement due to the voting approach.

Unfortunately, Expression (1) is not easily amenable to a closed form. However, numerical calculations of $\phi(m)$ have been carried out and are summarized in Figure 3 for a specific illustration: A $(4, 4)$ probing scheme, i.e., $k = 8$. Although specific, this example is quite significant because it represents, in a sense, the accepted state of the art in terms of the microarray size. Figure 3 displays the probability of incorrect voting for Mode 1 failure as a function of ℓ (point of failure) and m (sequence length).

As expected, this function increases with m for fixed ℓ and tends to 1 as ℓ decreases, since in such a condition the tie $j = h = 0$ becomes the most

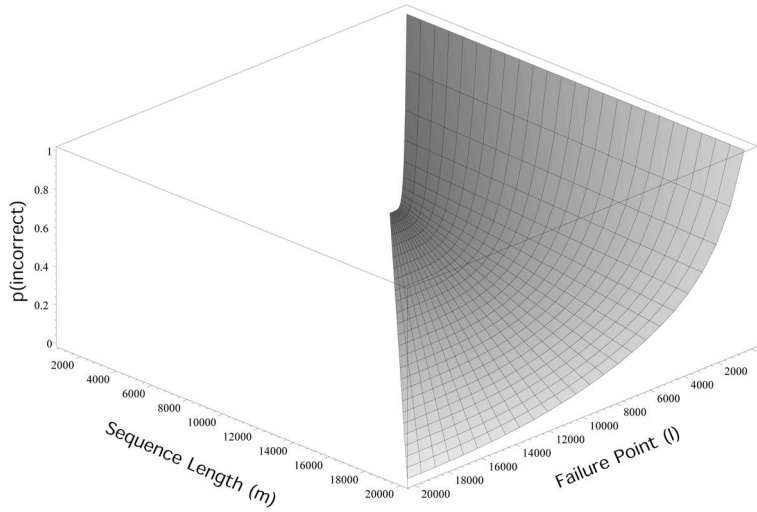


Fig. 3. Probability of incorrect voting for Mode 1 failure as a function of ℓ and m ($\ell \leq m$).

likely event. The quantities $P(\text{vote failure}|\text{Mode } i)$ are obtained by averaging the previous functions over the parameter ℓ. Finally, the quantities $P_1, P_2, P(\text{vote failure}|\text{Mode 1})$ and $P(\text{vote failure}|\text{Mode 2})$, each a function of m, are combined as in (2) to obtain the desired performance function $\phi(m)$. The latter is presented as $1 - \phi(m)$ in Figure 4, to be contrasted with the probability of success of the previous algorithm [PU00] without the voting provision.

Shown in the same figure are sets of experimental data points for the two situations, which are in remarkable agreement with the corresponding analyses. At 90%-confidence the length of the correctly reconstructible sequence increases by nearly 20%, which quantifies the enhancement afforded by the approach discussed in this paper, quite likely at the frontier of achievable performance.

References

[BS91] W. Bains and G.C. Smith, A novel method for DNA sequence determination. *Jour. of Theoretical Biology*(1988), 135, 303-307.

[DFS94] M.E.Dyer, A.M.Frieze, and S.Suen, The probability of unique solutions of sequencing by hybridization. *Journal of Computational Biology*, 1 (1994) 105-110.

[D89] R. Drmanac, I. Labat, I. Bruckner, and R. Crkvenjakov, Sequencing of megabase plus DNA by hybridization. *Genomics*,(1989),4, 114-128.

[LB94] D. Loakes and D.M. Brown, 5-Nitroindole as a universal base analogue. *Nucleic Acids Research*,(1994), 22, 20,4039-4043.

[L88] Yu.P. Lysov, V.L. Florentiev, A.A. Khorlin, K.R. Khrapko, V.V. Shih, and A.D. Mirzabekov, Sequencing by hybridization via oligonucleotides. A novel method. *Dokl. Acad. Sci. USSR*,(1988) 303, 1508-1511.

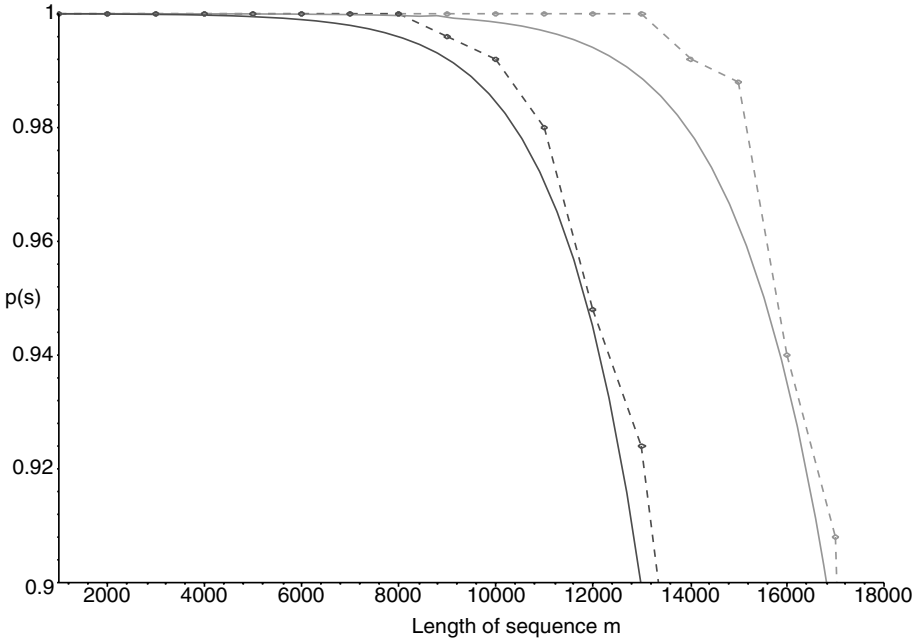


Fig. 4. Comparison of the algorithms without (left) and with (right) voting provision for a $(4, 4)$ probing scheme (probability of success versus m). Analytical curves for the two cases are paired with corresponding extensive simulation data.

[P89] P.A.Pevzner, l-tuple DNA sequencing: computer analysis. *Journ. Biomolecul. Struct. & Dynamics* (1989) 7, 1, 63-73.

[P91] P.A.Pevzner, Yu.P. Lysov, K.R. Khrapko, A.V. Belyavsky, V.L. Florentiev, and A.D. Mirzabekov, Improved chips for sequencing by hybridization. *Journ. Biomolecul. Struct. & Dynamics* (1991) 9, 2, 399-410.

[PFU99] F.P. Preparata, A.M. Frieze, E. Upfal. On the Power of Universal Bases in Sequencing by Hybridization. *Third Annual International Conference on Computational Molecular Biology*. April 11 - 14, 1999, Lyon, France, pp. 295–301.

[PU00] F.P. Preparata and E. Upfal. Sequencing-by-Hybridization at the information-theory bound: An optimal algorithm. *Journal of Computational Biology*, 7, 3/4, 621-630 (2000).

[S96] E.M. Southern, DNA chips: analysing sequence by hybridization to oligonucleotide on a large scale, *Trends in Genetics*, (1996) 12,3,110-115.

Non-approximability of Weighted Multiple Sequence Alignment

Bodo Siebert*

Institut für Theoretische Informatik
Med. Universität zu Lübeck
Wallstraße 40, 23560 Lübeck, Germany
siebert@tcs.mu-luebeck.de

Abstract. We consider a weighted generalization of multiple sequence alignment with sum-of-pair score. Multiple sequence alignment without weights is known to be $\mathcal{NP}$-complete and can be approximated within a constant factor, but it is unknown whether it has a polynomial time approximation scheme. Weighted multiple sequence alignment can be approximated within a factor of $O(\log^2 n)$ where n is the number of sequences.
We prove that weighted multiple sequence alignment is MAX $\mathcal{SNP}$-hard and establish a numerical lower bound on its approximability, namely $\frac{324}{323} - \epsilon$. This lower bound is obtained already for the simple binary weighted case where the weights are restricted to 0 and 1. Furthermore, we show that weighted multiple sequence alignment and its restriction to binary weights can be approximated exactly to the same degree.

1 Introduction

Multiple sequence alignment (MSA) is an important problem in computational biology. The alignment of a group of protein or nucleotide sequences yields information about the relationships between these sequences and it is also used to detect similarities (so called "homologous regions") between them. This information is applied in constructing evolutionary trees and finding coherences between the function and structure of proteins and their sequences.

Many objective functions have been suggested to measure the quality of a multiple sequence alignment. One of the most widely used is the so called sum-of-pair score (SP-score, Carrillo et al. [6]).

MSA with SP-score is known to be $\mathcal{NP}$-complete (Wang et al. [12]). For the case that the scoring function does not have to be a metric, Just has shown that MSA with SP-score is MAX $\mathcal{SNP}$-hard [9]. Akutsu et al. have investigated the multiple sequence alignment problem under several scoring functions, namely $\#LOG\#$-score and IC-score [1]. They have shown that a variant of the multiple sequence alignment problem called local multiple alignment is MAX $\mathcal{SNP}$-hard under these scoring schemes.

* supported by DFG research grant RE 672/3

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 75–85, 2001.
© Springer-Verlag Berlin Heidelberg 2001

However, if the scoring function fulfils the triangle inequality, no lower bound for this problem is known so far. The complexity of MSA over an alphabet of fixed size with metric SP-scoring functions is of main interest. According to Jiang et al. the approximability of MSA with metric SP-score is an important open problem in computational biology [8].

To represent existing knowledge about the relationships of the sequences considered, a weighted variant of MSA was introduced by Wu et al. [13]. Each pair of sequences is assigned a nonnegative value reflecting their degree of relationship. This means that a pair which is assumed to be closely related will be assigned a high weight while a less related pair will be assigned a smaller weight. This generalization of MSA is called weighted MSA, or WMSA for short.

In this paper we also examine a restricted version of WMSA called binary weighted MSA (BMSA), where the weights are restricted to 0 and 1. The binary weights can be used to represent an arbitrary graph over which multiple sequence alignments can be determined. We will prove that BMSA is equivalent to WMSA with respect to their approximability. Thus, an approximation algorithm for BMSA directly yields an approximation algorithm for the general case with the same performance ratio. Moreover, we prove the MAX $\mathcal{SNP}$-hardness and a numerical lower bound for the approximability of BMSA. These results are obtained even if the sequences are of fixed length and the alphabet is of fixed size. Thus, the difficulty of multiple sequence alignment is caused by the number of sequences, not by their length.

In the next section we give a formal definition of the problems considered. The reduction from WMSA to BMSA is presented in section 3. In section 4 we prove a lower bound for the approximability of a problem called MAX-E2-neg-Lin2. This result will be used in section 5 to prove a lower bound for the approximability of BMSA.

2 Definitions and Notations

Let Σ be an alphabet and $\Sigma' := \Sigma \cup \{-\}$, where "$-$" denotes a gap symbol. $S[l]$ denotes the l-th symbol of a sequence S. Let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a family (a multiset) of sequences over Σ. An *alignment* of $\mathcal{S}$ is a family $\mathcal{A} = \{\tilde{S}_1, \ldots, \tilde{S}_n\}$ of sequences over Σ' such that all $\tilde{S}_i$ have equal length and $\tilde{S}_i$ is obtained from S_i by inserting gaps. The following is an example of an alignment of three sequences $ATTCTG$, $TTCTTTG$ and $ATTGTT$.

$$\begin{array}{l} ATTCT{-}{-}G \\ {-}TTCTTTG \\ ATTGT{-}T{-} \end{array}$$

A function $d : {\Sigma'}^2 \to \mathbb{N}$ will be called *scoring function* if it is a metric, i.e. for any $x, y, z \in \Sigma'$ we have $d(x, y) = 0$ iff $x = y$, $d(x, y) = d(y, x)$, and $d(x, z) \leq d(x, y) + d(y, z)$. We define the distance of two sequences $\tilde{S}_i$ and $\tilde{S}_j$ of length l as $D(\tilde{S}_i, \tilde{S}_j) := \sum_{k=1}^{l} d(\tilde{S}_i[k], \tilde{S}_j[k])$.

Carrillo and Lipman introduced a scoring scheme for alignments called *sum-of-pair score* (SP-score, [6]). The SP-score of an alignment $\mathcal{A} = \{\tilde{S}_1, \ldots, \tilde{S}_n\}$ is defined by $D(\mathcal{A}) := \sum_{1 \le i < j \le n} D(\tilde{S}_i, \tilde{S}_j)$. Multiple sequence alignment (MSA) is the problem of finding an alignment with minimum SP-score.

Wu et al. generalized MSA to weighted sum-of-pair score [13]. The weights are given by $W := (w_{S_i,S_j})_{S_i,S_j \in \mathcal{S}}$, a symmetric matrix of nonnegative integers. Then the weighted SP-score of an alignment $\mathcal{A}$ is $D_W(\mathcal{A}) := \sum_{1 \le i < j \le n} w_{S_i,S_j} \cdot D(\tilde{S}_i, \tilde{S}_j)$. This generalization is called *weighted multiple sequence alignment* (WMSA). The aim is to find an alignment with minimum weighted SP-score.

An instance of WMSA is a 4-tuple $(\Sigma, \mathcal{S}, d, W)$. We consider the case of a fixed alphabet Σ and a fixed scoring function d. Thus, a problem instance of WMSA is given by a pair $(\mathcal{S}, W)$. It is easy to see that any lower bound for this case also holds if we allow arbitrary scoring functions and alphabets.

A special case of WMSA is *binary weighted MSA* (BMSA), where the weights are restricted to 0 and 1.

It has been shown that MSA with SP-score is $\mathcal{NP}$-complete [12]. For an arbitrary fixed constant r, MSA can be approximated in polynomial time within a factor of $2 - \frac{r}{n}$, where $n \ge r$ is the number of sequences [4]. It is unknown whether MSA admits a polynomial time approximation scheme (PTAS, see e.g. Ausiello et al. [3]). WMSA with arbitrary weights can be approximated within a factor of $O(\log^2 n)$ [13]. Using a technique of Bartal [5] one can obtain a randomized $O(\log n \cdot \operatorname{llog} n)$ approximation.

Papadimitriou et al. introduced a class of optimization problems called MAX $\mathcal{SNP}$ [10]. They showed that there exist problems which are MAX $\mathcal{SNP}$-complete with respect to L-reductions. In the following, $\operatorname{opt}(I)$ denotes the optimal score of an instance I of an optimization problem. For example, $\operatorname{opt}(\mathcal{S})$ denotes the score of an optimal (weighted) alignment of $\mathcal{S}$.

Definition 1. *Let Π and Π' be two optimization problems. Then Π L-reduces to Π' if there exist polynomial time computable functions f_1, f_2 and constants $\gamma_1, \gamma_2 > 0$ such that for each instance I of Π:*

1. *Function f_1 produces an instance $I' = f_1(I)$ of Π' such that $\operatorname{opt}(I') \le \gamma_1 \cdot \operatorname{opt}(I)$.*
2. *Given a solution S' of I' with cost $c'(S')$, function f_2 produces a solution $S = f_2(I, S')$ of I with cost $c(S)$ such that $|c(S) - \operatorname{opt}(I)| \le \gamma_2 \cdot |c'(S') - \operatorname{opt}(I')|$.*

No MAX $\mathcal{SNP}$-hard problem has a PTAS, unless $\mathcal{NP} = \mathcal{P}$ (Arora et al. [2]).

3 Reduction from WMSA to BMSA

Let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a family of sequences over Σ and $W = (w_{S_i,S_j})$ be a weight matrix. Let l be the maximal length of the sequences in $\mathcal{S}$ and $d_{\max}$ be the maximum of the scoring function d. We assume that the weights and the scoring function are unary coded. This does not seem to be a restriction because in practice the weights are very small and the scoring function is fixed.

We construct a family of sequences $\mathcal{S}'$ as an instance of BMSA as follows. Let $K := 2 \cdot d_{\max} \cdot l$. For a sequence $S_j \in \mathcal{S}$ generate K copies $T_j^k \in \mathcal{S}'$ $(1 \le k \le K)$ of this sequence. Furthermore, for each $1 \le i \le n$ construct w_{S_i,S_j} copies $S_j^{i,\mu} \in \mathcal{S}'$ $(1 \le \mu \le w_{S_i,S_j})$ of S_j. The weight matrix $W' = (w'_{I,J})_{I,J \in \mathcal{S}'}$ is given by

$$w'_{I,J} := \begin{cases} 1 & \text{if } I \equiv S_j^{i,\mu} \text{ and } J \equiv S_i^{j,\mu}\,, \\ 1 & \text{if } I \equiv S_j^{i,\mu} \text{ and } J \equiv T_j^k \text{ or vice versa}\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

where $A \equiv B$ means that A and B are not only equal but denote the same sequence.

Since the weights and the scoring function are unary coded, the input size N of the instance of WMSA fulfils the bound $N \in \Omega\big(n \cdot l + \sum_{i,j=1}^{n} w_{S_i,S_j}\big)$. On the other hand, the input size N' of the constructed instance of BMSA satisfies

$$N' \in O\big(\underbrace{n \cdot K \cdot l}_{T_\cdot^\cdot} + \underbrace{l \cdot \textstyle\sum_{i,j=1}^{n} w_{S_i,S_j}}_{S_\cdot^{i,\cdot}} + \underbrace{(n \cdot K + \textstyle\sum_{i,j=1}^{n} w_{S_i,S_j})^2}_{W'}\big)\,.$$

Note that N' is polynomially bounded by N.

Lemma 1. *If $\mathcal{S}$ has an alignment $\mathcal{A}$ with weighted score $D_W(\mathcal{A})$ then $\mathcal{S}'$ has an alignment $\mathcal{A}'$ with score $D_{W'}(\mathcal{A}') = D_W(\mathcal{A})$.*

Proof. Let $\mathcal{A} = \{\tilde{S}_1, \ldots, \tilde{S}_n\}$ be an alignment of $\mathcal{S}$ with weighted score D. We obtain an alignment $\mathcal{A}' = \{\tilde{A} | A \in \mathcal{S}'\}$ of $\mathcal{S}'$ by setting $\tilde{T}_j^k = \tilde{S}_j$ and $\tilde{S}_j^{i,\mu} = \tilde{S}_j$ for all j, k, i, μ. The score of $\mathcal{A}'$ with respect to the weight matrix W' is

$$D_{W'}(\mathcal{A}') = \sum_{i,j=1}^{n} \sum_{\mu=1}^{w_{S_i,S_j}} \sum_{k=1}^{K} \underbrace{D(\tilde{S}_j^{i,\mu}, \tilde{T}_j^k)}_{=0} + \sum_{1 \le i < j \le n} \sum_{\mu=1}^{w_{S_i,S_j}} \underbrace{D(\tilde{S}_i^{j,\mu}, \tilde{S}_j^{i,\mu})}_{=D(\tilde{S}_i,\tilde{S}_j)} = D_W(\mathcal{A})\,.$$

□

Lemma 2. *Given an alignment $\mathcal{A}'$ of $\mathcal{S}'$ with weighted score $D_{W'}(\mathcal{A}')$ we can construct an alignment $\mathcal{A}$ of $\mathcal{S}$ with less or equal score in polynomial time.*

Proof. Let $\mathcal{A}' = \{\tilde{A} | A \in \mathcal{S}'\}$ be an arbitrary alignment of $\mathcal{S}'$ with score $D_{W'}(\mathcal{A}')$. The copies of a sequence $S_j \in \mathcal{S}$ will be called *consistent* if there exists a sequence B_j with $\tilde{T}_j^k = B_j$ and $\tilde{S}_j^{i,\mu} = B_j$ for all k, i, μ. The sequence B_j is called *block*.

We consider the case that for some j_0 the copies of S_{j_0} are not consistent and distinguish two cases. First, if not all $\tilde{T}_{j_0}^k$ are equal, let

$D_k := \sum_{i=1}^{n} \sum_{\mu=1}^{w_{S_i,S_j}} D(\tilde{T}_{j_0}^k, \tilde{S}_{j_0}^{i,\mu})$

be the score of $\tilde{T}_{j_0}^k$ with the sequences $\tilde{S}_{j_0}^{i,\cdot}$. Choose k_0 such that D_{k_0} is minimal among all D_k and set $\tilde{T}_{j_0}^k = \tilde{T}_{j_0}^{k_0}$ for all $k \ne k_0$. This way we obtain a new alignment with less or equal score.

Now we consider the case that there exists a B_{j_0} such that $\tilde{T}^k_{j_0} = B_{j_0}$ for all k. Then there exists a sequence $\tilde{S}^{i_0,\mu_0}_{j_0} \neq B_{j_0}$. This sequence yields at least score K with the sequences $\tilde{T}_{j_0}$, because it yields a score of at least 1 with every $\tilde{T}^k_{j_0}$. Set $\tilde{S}^{i_0,\mu_0}_{j_0} = B_{j_0}$. Then $\tilde{S}^{i_0,\mu_0}_{j_0}$ yields score 0 with any $\tilde{T}^k_{j_0}$ and at most score K with $\tilde{S}^{j_0,\mu_0}_{i_0}$. Thus, the new alignment has less or equal score.

By these modifications we iteratively obtain a new alignment of $\mathcal{S}'$ such that for any $j \in \{1,\ldots,n\}$ the copies of S_j are consistent with block B_j. The blocks of $\mathcal{S}'$ induce an alignment $\mathcal{A} = \{B_1,\ldots,B_n\}$ of $\mathcal{S}$ with score $D_W(\mathcal{A}) = \sum_{1\leq i<j\leq n} w_{S_i,S_j} \cdot D(B_i,B_j) \leq D_{W'}(\mathcal{A}')$. □

With these results we have shown that a λ-approximation for BMSA can be used as a λ-approximation for WMSA. Thus, the following theorem holds.

Theorem 1. *If* BMSA *can be approximated within a constant factor λ in polynomial time, then* WMSA *can also be approximated within λ in polynomial time.* □

4 The Non-approximability of MAX-E2-neg-Lin2

We consider the multiplicative group $\{1,-1\}$. Let $\mathcal{G} = \{G_1,\ldots,G_t\}$ be a multiset of linear equations over the variables $U = \{x_1,\ldots,x_r\}$, $G_i \widehat{=} x_{\alpha_{i,1}} \cdot \ldots \cdot x_{\alpha_{i,k}} = a_i$, $k \geq 2$, $\alpha_{i,q} \in \{1,\ldots,r\}$, and $a_i \in \{1,-1\}$ is a constant. MAX-Ek-Lin2 is the optimization problem of finding the maximum number of simultaneously satisfiable equations. A restriction of MAX-Ek-Lin2 is MAX-Ek-neg-Lin2, where $a_i = -1$ for all $1 \leq i \leq t$.

MAX-E2-neg-Lin2 is exactly the problem MAX-Cut (see e.g. [3]) where the equations correspond to the edges, the variables correspond to the nodes, and multiple edges are allowed. Therefore, MAX-E2-neg-Lin2 is MAX $\mathcal{SNP}$-complete [10]. We use MAX-E2-neg-Lin2 here due to the simpler notation.

An instance of MAX-Ek-Lin2 or MAX-Ek-neg-Lin2 consisting of t equations will be called *η-satisfiable* iff $\eta \cdot t$ is the maximum number of simultaneously satisfiable equations. Håstad proved in [7] that it is $\mathcal{NP}$-hard to distinguish $\left(1-\epsilon\right)$-satisfiable and $\left(\frac{1}{2}+\epsilon\right)$-satisfiable instances of MAX-E3-Lin2 for any $\epsilon > 0$.

Instead of the known lower bound for the approximability of MAX-Cut (Håstad [7] and Trevisan et al. [11]) we will construct a reduction from MAX-E3-Lin2 to MAX-E2-neg-Lin2 to prove that it is $\mathcal{NP}$-hard to distinguish $\left(\frac{18}{22}-\epsilon\right)$- and $\left(\frac{17}{22}+\epsilon\right)$-satisfiable instances of MAX-E2-neg-Lin2 for any $\epsilon > 0$; the gadget used by Trevisan et al. [11] does not yield such a gap directly. This result will be used in section 5 to establish the lower bound for the approximability of BMSA.

We will now reduce MAX-E3-Lin2 to MAX-E2-neg-Lin2. Let $\mathcal{G}=\{G_1,\ldots,G_t\}$ be a multiset of equations over variables U, $G_i \widehat{=} x_{\alpha_{i,1}} \cdot x_{\alpha_{i,2}} \cdot x_{\alpha_{i,3}} = a_i$.

We construct an instance $\mathcal{G}'$ of MAX-E2-neg-Lin2 with $22 \cdot t$ equations and $4 \cdot t + 2 \cdot r + 2$ variables. The reduction is similar to the reduction from MAX-E3-Lin2 to MAX-E2-Lin2 in [7]. The set of variables U' is given by

$$U' = \{x_j^+, x_j^- | 1 \leq j \leq r\} \cup \{z^+, z^-\} \cup \{p_{i,1}, p_{i,2}, p_{i,3}, p_{i,z} | 1 \leq i \leq t\}\,.$$

Note that if an assignment satisfies an equation of an instance of MAX-E2-neg-Lin2, then the negated assignment also satisfies the equation. So without loss of generality we assume that in any case $z^+ = 1$.

We interpret $x_j^+ = x_j$. We call an assignment *consistent for* x_j if $x_j^+ \neq x_j^-$ and therefore $x_j^+ = x_j = (-x_j^-)$. An assignment that is consistent for every x_j and where $z^+ \neq z^-$ is called *consistent*.

For an equation $G_i \widehat{=} x_{\alpha_{i,1}} \cdot x_{\alpha_{i,2}} \cdot x_{\alpha_{i,3}} = a_i$ we construct the twelve equations

$$\begin{aligned} x^+_{\alpha_{i,q}} \cdot p_{i,q'} &= -1 \quad \text{for } q, q' = 1,2,3 \text{ and } q \neq q' \,, \\ x^+_{\alpha_{i,q}} \cdot p_{i,z} &= -1 \quad \text{for } q = 1,2,3 \,, \\ x^-_{\alpha_{i,q}} \cdot p_{i,q} &= -1 \quad \text{for } q = 1,2,3 \,. \end{aligned}$$

We add either the four equations $z^+ \cdot p_{i,q} = -1$ $(q = 1,2,3)$ and $z^- \cdot p_{i,z} = -1$ if $a_i = 1$ or the four equations $z^- \cdot p_{i,q} = -1$ $(q = 1,2,3)$ and $z^+ \cdot p_{i,z} = -1$ if $a_i = -1$. For every equation in $\mathcal{G}$ we construct the three equations $x^+_{\alpha_{i,q}} \cdot x^-_{\alpha_{i,q}} = -1$ $(q = 1,2,3)$. Finally, we add the equation $z^+ \cdot z^- = -1$ three times. Note that $\mathcal{G}'$ contains $3 \cdot t$ times the equation $z^+ \cdot z^- = -1$. Let n_j be the number of occurrences of the variable x_j in $\mathcal{G}$. Then $\mathcal{G}'$ contains n_j times the equation $x_j^+ \cdot x_j^- = -1$.

For every equation $G_i \in \mathcal{G}$ we have constructed 22 equations for $\mathcal{G}'$. These 22 equations are called the *representation of* G_i.

Lemma 3. *Let an arbitrary assignment for U be given. Assign $z^- = -1$ and $x_j^+ = x_j$, $x_j^- = (-x_j)$ for $j = 1, \ldots, r$. Then for any $i \in \{1, \ldots, t\}$ there exists an assignment for $p_{i,1}$, $p_{i,2}$, $p_{i,3}$, and $p_{i,z}$ such that 18 equations of the representation of G_i are satisfied if G_i is satisfied by the given assignment and 16 equations of the representation are satisfied if G_i is not satisfied.*

It is not possible to satisfy more than 18 equations of the representation if G_i is satisfied by the assignment and to satisfy more than 16 equations if G_i is not satisfied by the assignment.

Proof. The lemma can be proved by testing all possible assignments. □

If an assignment for U satisfies g of the t equations of $\mathcal{G}$, then the corresponding consistent assignment for U' satisfies $16 \cdot t + 2 \cdot g$ equations of $\mathcal{G}'$. This assignment can be found efficiently by adjusting the assignment for $p_{i,1}$, $p_{i,2}$, $p_{i,3}$, and $p_{i,z}$. On the other hand, a consistent assignment for U' that satisfies $16 \cdot t + 2 \cdot g$ equations of $\mathcal{G}'$ yields an assignment for U that satisfies g equations of $\mathcal{G}$.

Lemma 4. *Given an arbitrary assignment for U' that satisfies $16 \cdot t + 2 \cdot g$ equations of $\mathcal{G}'$, a consistent assignment that satisfies at least this amount of equations of $\mathcal{G}'$ can be computed in polynomial time.*

Proof. First assume that $z^+ = z^-$ in the given assignment. Then the $3 \cdot t$ equations $z^+ \cdot z^- = -1$ are not satisfied by the assignment. Let $z^- = (-z^+)$. Then these $3 \cdot t$ equations will be satisfied. On the other hand, z^- occurs in only $3 \cdot t$

other equations. Thus, at most $3 \cdot t$ equations are no longer satisfied. Altogether the number of satisfied equations is not decreased by this modification.

If there exists a j with $x_j^+ = x_j^-$, then there are n_j equations $x_j^+ \cdot x_j^- = -1$ that are not satisfied by the assignment. Let $x_j^- = (-x_j^+)$. Then the n_j equations $x_j^+ \cdot x_j^- = -1$ are satisfied by the modified assignment. On the other hand x_j^- occurs in only n_j other equations. Thus, at most n_j equations are no longer satisfied. The number of satisfied equations is thus not decreased by this modification.

This way we iteratively obtain a consistent assignment. Obviously, the modifications can be computed in polynomial time. □

Now we can prove the following theorem used in section 5.

Theorem 2. *For any $\epsilon > 0$ it is $\mathcal{NP}$-hard to distinguish $\left(\frac{18}{22} - \epsilon\right)$- and $\left(\frac{17}{22} + \epsilon\right)$-satisfiable instances of* MAX-E2-neg-Lin2.

Proof. An instance of MAX-E3-Lin2 is η-satisfiable iff the corresponding instance of MAX-E2-neg-Lin2 is $\left(\frac{16+2\cdot\eta}{22}\right)$-satisfiable. According to Håstad [7] it is $\mathcal{NP}$-hard to distinguish $\left(1 - \xi\right)$- and $\left(\frac{1}{2} + \xi\right)$-satisfiable instances of MAX-E3-Lin2 for any $\xi > 0$. Thus, it is $\mathcal{NP}$-hard to distinguish $\left(\frac{16+2\cdot(1-\xi)}{22}\right)$- and $\left(\frac{16+2\cdot(\frac{1}{2}+\xi)}{22}\right)$-satisfiable instances of MAX-E2-neg-Lin2. Choosing $\xi = 11 \cdot \epsilon$ completes the proof. □

Since MAX-Cut and MAX-E2-neg-Lin2 are exactly the same problem, we obtain the same approximability gap for MAX-Cut.

Corollary 1. *For any $\epsilon > 0$ it is $\mathcal{NP}$-hard to decide whether the maximum cut of an instance $G = (V, E)$ (where multiple edges are allowed) of* MAX-Cut *consists of at most $\left(\frac{17}{22} + \epsilon\right) \cdot |E|$ or at least $\left(\frac{18}{22} - \epsilon\right) \cdot |E|$ edges.* □

5 The Non-approximability of BMSA

In this section we reduce MAX-E2-neg-Lin2 to BMSA. Let $\mathcal{G} = \{G_1, \ldots, G_t\}$ be an instance of MAX-E2-neg-Lin2 over a set of variables $U = \{x_1, \ldots, x_r\}$, $G_i \widehat{=} x_{\alpha_{i,1}} \cdot x_{\alpha_{i,2}} = -1$, $\alpha_{i,q} \in \{1, \ldots, r\}$. We construct a family of sequences $\mathcal{S} = \{Z\} \cup \{X_j | j = 1, \ldots, r\} \cup \{Y_{i,1}, Y_{i,2} | i = 1, \ldots, t\}$ over the alphabet $\Sigma = \{\bullet, \circ, \times\}$. Let $Z := \circ\circ\circ\circ\circ\circ\circ\circ$ be a sequence of length 8. Z will be used as a control sequence. For $j \in \{1, \ldots, r\}$ let $X_j := \bullet\circ\circ\circ\circ\circ\circ\circ\bullet$ be a sequence of length 9 that represents the variable $x_j \in U$. For each $i \in \{1, \ldots, t\}$ create two sequences $Y_{i,1} := \bullet\circ\circ\times\circ\times\circ\circ\bullet$ and $Y_{i,2} := \bullet\circ\circ\circ\times\circ\circ\circ\bullet$, each of length 9. $Y_{i,q}$ represents the variable $x_{\alpha_{i,q}}$ in G_i.

The scoring function is given in the following table. Note that it is a metric.

	-	•	○	×
-	0	1	2	5
•	1	0	1	4
○	2	1	0	3
×	5	4	3	0

The weight matrix $W = (w_{I,J})_{I,J \in \mathcal{S}}$ is given by

$$w_{I,J} := \begin{cases} 1 & \text{if } I \equiv Y_{i,q} \text{ and } J \equiv Y_{i,q'}\,, \\ 1 & \text{if } I \equiv Z \text{ and } J \equiv Y_{i,q} \text{ or vice versa}\,, \\ 1 & \text{if } I \equiv Y_{i,q} \text{ and } J \equiv X_{\alpha_{i,q}} \text{ or vice versa}\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

The set $\mathcal{S}_i = \{Y_{i,1}, Y_{i,2}, X_{\alpha_{i,1}}, X_{\alpha_{i,2}}\}$ will be called the *representation of* G_i. Note that in general a sequence X_j occurs in more than one representation.

Let $\mathcal{A} = \{\tilde{S} | S \in \mathcal{S}\}$ be an alignment of $\mathcal{S}$. Then $D_i(\mathcal{A})$ denotes the score of the equation G_i, $D_i(\mathcal{A}) = D(\tilde{Y}_{i,1}, \tilde{Y}_{i,2}) + D(\tilde{Y}_{i,1}, \tilde{X}_{\alpha_{i,1}}) + D(\tilde{Y}_{i,2}, \tilde{X}_{\alpha_{i,2}}) + D(\tilde{Y}_{i,1}, \tilde{Z}) + D(\tilde{Y}_{i,2}, \tilde{Z})$. By the construction of the weight matrix, $D_W(\mathcal{A}) = \sum_{i=1}^{t} D_i(\mathcal{A})$ holds.

Definition 2. *An alignment* $\mathcal{A} = \{\tilde{S} | S \in \mathcal{S}\}$ *of* $\mathcal{S}$ *will be called* variable-consistent *with respect to an assignment for* U *if, after eliminating all columns consisting solely of gaps (which do not affect the score), the following holds for all* j, i, *and* q*:*

1. $\tilde{Z} = -Z-$
2. $\tilde{X}_j = \begin{cases} X_j- & \text{if } x_j = -1 \\ -X_j & \text{if } x_j = 1 \end{cases}$
3. $\tilde{Y}_{i,q} = \begin{cases} Y_{i,q}- & \text{if } x_{\alpha_{i,q}} = -1 \\ -Y_{i,q} & \text{if } x_{\alpha_{i,q}} = 1 \end{cases}$

The following lemma follows immediately from this definition.

Lemma 5. *An alignment is variable-consistent iff for all* $i = 1, \ldots, t$ *and* $q = 1, 2$ *the following properties hold:*

A. *Either* $Y_{i,q}[1]$ *or* $Y_{i,q}[9]$ *matches a gap in* Z*. No other character of* Z *or* $Y_{i,q}$ *matches a gap in the other sequence.*
B. *No character in either of the two sequences* $Y_{i,q}$, $X_{\alpha_{i,q}}$ *matches a gap in the other sequence.* □

These properties are referred to as property A and B. The following is an example of a variable-consistent alignment representing the equation $G_i \mathrel{\widehat{=}} x_1 \cdot x_2 = -1$ which is satisfied by $x_1 = -1$ and $x_2 = 1$.

$$\begin{array}{rcl} \tilde{Y}_{i,1} & = & \bullet\ \circ\ \circ\ \times\ \circ\ \times\ \circ\ \circ\ \bullet\ - \\ \tilde{Y}_{i,2} & = & -\ \bullet\ \circ\ \circ\ \circ\ \times\ \circ\ \circ\ \circ\ \bullet \\ \tilde{X}_1 & = & \bullet\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \bullet\ - \\ \tilde{X}_2 & = & -\ \bullet\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \bullet \\ \tilde{Z} & = & -\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ \circ\ - \end{array}$$

Note the functional region of a pair $Y_{i,1}, Y_{i,2}$ given by the triples $\times\circ\times$ and $\circ\times\circ$. If $Y_{i,1}$ and $Y_{i,2}$ represent the same value, the functional region yields a weighted score of 9. Otherwise, it yields a weighted score of 3. If an alignment $\mathcal{A}$ is variable-consistent, we have $D_i(\mathcal{A}) = 29$ if G_i is satisfied by the represented assignment and $D_i(\mathcal{A}) = 31$ otherwise.

The next two lemmas have similar proofs. Thus, we only give a proof of the first.

Lemma 6. *Alignments of the pairs* $\{Y_{i,1}, Z\}$ *and* $\{Y_{i,2}, Z\}$ *yield scores of* 8 *and* 5*, respectively, if they fulfil property A. Violating property A yields scores of at least* 10 *and* 7*, respectively.*

Proof. An alignment of $\{Y_{i,1}, Z\}$ that fulfils property A yields score 8.

Let us consider an alignment of $\{Y_{i,1}, Z\}$ that does not fulfil property A. Then at least one of the characters $Y_{i,1}[2], \ldots, Y_{i,1}[8], Z[1], \ldots, Z[8]$ matches a gap in the other sequence.

We distinguish two cases. If there is an "×" in $Y_{i,1}$ matching a gap in Z, then the alignment yields a score of 5 for this "×" plus 3 for the other "×" plus 1 for each "•". So altogether it yields a score of at least 10.

On the other hand consider the case that no "×" in $Y_{i,1}$ matches a gap in Z. Then there is a "○" in $Y_{i,1}$ or Z matching a gap in the other sequence. So the alignment yields a score of 3 for each "×" plus 1 for each "•" plus 2 for the "○" matching a gap. So the alignment again yields a score of at least 10.

The statement about $Y_{i,2}$ and Z can be proved in a similar fashion. □

Lemma 7. *Alignments of the pairs* $\{Y_{i,1}, X_{\alpha_{i,1}}\}$ *and* $\{Y_{i,2}, X_{\alpha_{i,2}}\}$ *yield scores of* 6 *and* 3*, respectively, if they fulfil property B. Violating property B yields scores of at least* 8 *and* 5*, respectively.* □

With the fact that an optimal alignment of a pair $\{Y_{i,1}, Y_{i,2}\}$ has score 7 we can prove the following.

Lemma 8. *Given an arbitrary alignment with score* $31 \cdot t - 2 \cdot g$ *we can construct a variable-consistent alignment with less or equal score in polynomial time.*

Proof. Let $\mathcal{A}$ be an arbitrary alignment with $D_W(\mathcal{A}) = 31 \cdot t - 2 \cdot g$.

Let I be the set of all i such that $Y_{i,1}$ and $Y_{i,2}$ fulfil properties A and B. This implies an assignment for the variables $U_I = \{x_j \in U | \exists i \in I : X_j \in \mathcal{S}_i\}$. Let $\overline{I} = \{1, \ldots, t\} \setminus I$. Because in every set $\mathcal{S}_i$ for $i \in \overline{I}$ there exists a sequence $Y_{i,q}$ that violates property A or B, we have $D_i(\mathcal{A}) \geq 31$ for each $i \in \overline{I}$ due to Lemmas 6 and 7.

For $i \in \overline{I}$ if $x_{\alpha_{i,q}} \in U_I$ $(q \in \{1, 2\})$, we realign $Y_{i,q}$ with respect to $x_{\alpha_{i,q}}$. Then we assign an arbitrary value to the variables in $U \setminus U_I$ and realign the corresponding $Y_{i,q}$ and X_j.

By these modifications we obtain an alignment $\mathcal{A}'$. Then $D_i(\mathcal{A}') = D_i(\mathcal{A})$ for $i \in I$ and $D_i(\mathcal{A}') \leq 31 \leq D_i(\mathcal{A})$ otherwise. Thus, $D_W(\mathcal{A}') \leq D_W(\mathcal{A})$. $\mathcal{A}'$ is variable-consistent due to its construction and can be computed in polynomial time. □

The alignment obtained yields an assignment that satisfies at least g equations of $\mathcal{G}$.

Theorem 3. BMSA *is* MAX $\mathcal{SNP}$*-hard.*

Proof. We reduce MAX-E2-neg-Lin2 to BMSA. f_1 is given by the construction of $\mathcal{S}$ from a family $\mathcal{G}$ of t equations. One can see that $\text{opt}(\mathcal{S}) \leq 31 \cdot t$.

An equation of $\mathcal{G}$ will be satisfied by 2 of the 4 possible assignments of its variables. Therefore, for every multiset $\mathcal{G}$ of t equations an assignment exists that satisfies at least $\frac{1}{2} \cdot t$ equations. Then for $\gamma_1 = 62$ we have $\text{opt}(\mathcal{S}) \leq \gamma_1 \cdot \text{opt}(\mathcal{G})$.

Given an alignment of $\mathcal{S}$ with score $31 \cdot t - 2 \cdot g'$ for some g' we can find an assignment satisfying $g \geq g'$ equations of $\mathcal{G}$ due to Lemma 8. Let $\gamma_2 = \frac{1}{2}$, then $|g - \text{opt}(\mathcal{G})| \leq \gamma_2 \cdot |(31 \cdot t - 2 \cdot g') - \text{opt}(\mathcal{S})|$ holds. □

Theorem 4. BMSA *has no polynomial time approximation with performance ratio* $\frac{324}{323} - \epsilon$ *for any* $\epsilon > 0$*, unless* $\mathcal{NP} = \mathcal{P}$*.*

Proof. An instance of MAX-E2-neg-Lin2 consisting of t equations is η-satisfiable iff the corresponding instance of BMSA has an alignment with score $(31-2\cdot\eta)\cdot t$.

The optimal alignment of a BMSA instance corresponding to a $\left(\frac{18}{22} - \xi\right)$-satisfiable instance of MAX-E2-neg-Lin2 has score $\left(31-2\cdot\left(\frac{18}{22}-\xi\right)\right)\cdot t = \frac{323+22\cdot\xi}{11}\cdot t$. Using the $\left(\frac{324}{323} - \epsilon\right)$-approximation algorithm for BMSA we are able to find an alignment with score at most $\left(\frac{324}{323} - \epsilon\right) \cdot \frac{323+22\cdot\xi}{11} \cdot t =: K_1$.

The optimal alignment of a BMSA instance corresponding to a $\left(\frac{17}{22} + \xi\right)$-satisfiable instance of MAX-E2-neg-Lin2 has score $\left(31 - 2 \cdot \left(\frac{17}{22} + \xi\right)\right) \cdot t =: K_2$. We have $K_1 < K_2$ iff $\xi < \frac{1}{22} \cdot \frac{323^2 \cdot \epsilon}{647-323\cdot\epsilon}$. Choose ξ with $0 < \xi < \frac{1}{22} \cdot \frac{323^2 \cdot \epsilon}{647-323\cdot\epsilon}$. Then the $\left(\frac{324}{323} - \epsilon\right)$-approximation for BMSA can be used to distinguish $\left(\frac{18}{22} - \xi\right)$- and $\left(\frac{17}{22} + \xi\right)$-satisfiable instances of MAX-E2-neg-Lin2. This would imply $\mathcal{NP} = \mathcal{P}$ due to Theorem 2. □

Since WMSA is a generalization of BMSA it is also MAX $\mathcal{SNP}$-hard and we obtain the same non-approximability result.

6 Conclusions

We have shown MAX $\mathcal{SNP}$-hardness and proved a numerical lower bound for the approximability of weighted multiple sequence alignment (WMSA). These results hold even if we restrict the problem to binary weights (BMSA). Furthermore, BMSA and WMSA are equivalent with respect to their approximability. But the distance to the best known upper bound is huge. An obvious goal is to reduce this gap.

Finally, we would like to know how well the unweighted version of the multiple sequence alignment problem with metric SP-score can be approximated.

Acknowledgements

I wish to thank Martin Böhme, Andreas Jakoby, and Rüdiger Reischuk for valuable discussions and comments on this paper.

References

1. T. Akutsu, H. Arimura, and S. Shimozono. On approximation algorithms for local multiple alignment. In *Proc. 4th Ann. ACM Int. Conf. on Comput. Mol. Biol.*, pages 1–7, 2000.
2. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
3. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer-Verlag, 1999.
4. V. Bafna, E. L. Lawler, and P. A. Pevzner. Approximation algorithms for multiple sequence alignment. *Theor. Comp. Sc.*, 182(1–2):233–244, 1997.
5. Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proc. 30th Ann. Symp. Theor. Comput.*, pages 161–168. ACM, 1998.
6. H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48:1073–1082, 1988.
7. J. Håstad. Some optimal inapproximability results. In *Proc. 29th Ann. Symp. Theor. Comput.*, pages 1–10. ACM, 1997.
8. T. Jiang, P. Kearney, and M. Li. Some open problems in computational molecular biology. *SIGACT News*, 30(3):43–49, 1999.
9. W. Just. Computational complexity of multiple sequence alignment with SP-score. Technical report, Ohio University, 1999.
10. C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comp. Sys. Sc.*, 43(3):425–440, 1991.
11. L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson. Gadgets, approximation, and linear programming. *SIAM J. Comput.*, 29(6):2074–2097, 2000.
12. L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comp. Biol.*, 1(4):337–348, 1994.
13. B. Y. Wu, G. Lancia, V. Bafna, K. Chao, R. Ravi, and C. Y. Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM J. Comput.*, 29(3):761–778, 1999.

A Greedy Algorithm for Optimal Recombination

Shiquan Wu and Xun Gu*

Center of Bioinformatics and Biological Statistics
Iowa State University, Ames, IA 50011, USA
{sqwu,xgu}@iastate.edu

Abstract. Let Σ be an alphabet and Σ^n denote the collection of all sequences of length n over Σ. For any $\mathbf{s}_1 = a_1a_2\cdots a_ja_{j+1}\cdots a_n$, $\mathbf{s}_2 = b_1b_2\cdots b_j\ b_{j+1}\cdots b_n \in \Sigma^n$, a recombination of $\mathbf{s}_1$ and $\mathbf{s}_2$ at position j is defined as an operation that crosses $\mathbf{s}_1$ and $\mathbf{s}_2$ at position j and generates $\mathbf{t}_1 = a_1a_2\cdots a_jb_{j+1}\cdots b_n$ and $\mathbf{t}_2 = b_1b_2\cdots b_ja_{j+1}\cdots a_n$. Denote $\mathcal{A}$ and $\mathcal{S}$ two collections of sequences. In this paper, we discuss generating $\mathcal{A}$ from $\mathcal{S}$ by a series of recombinations in minimum number of steps. We present a greedy algorithm for finding the optimal recombination evolutionary history from $\mathcal{S}$ to any tree $\mathcal{A}$ of sequences when $|\mathcal{S}| = 2$.

1 Introduction

Various types of mutations on sequences play an important role in computational biology. Transformations using insertion, deletion, substitution, and reversal are widely studied by applying statistics and algorithms (cf. [1,2]). Recently, much attention has been paid to recombination of sequences. Hein developed some algorithms for recombination problems(cf. [3,4]). Kececioglu and Gusfield discussed a recombination distance problem on generating a third sequence from a given pair of sequences in optimal recombination cost (cf. [5]).

Generally, an edit distance problem of genomes is widely studied and its aim is to find the optimal evolutionary history from some ancestors to some descendents by using certain types of mutations such as insertion, deletion, substitution(point mutation),reversal, etc. Parsimony trees and phylogenetic trees are some intersting problems in the category. In [6], an alignment with recombination is discussed and it is an edit distance problem involved recombinations.

In this paper, we discuss a similar distance problem involved recombinations. The purpose is to generate a collection $\mathcal{A}$ of sequences from another collection $\mathcal{S}$ of sequences by a series of recombinations in minimum number of steps.

1.1 Problem and Example

Definition 1. *Let Σ be an alphabet (e.g., $\Sigma = \{a, c, g, t\}$, etc) and Σ^n the collection of all sequences of length n over Σ. For any $\mathbf{s}_1 = a_1a_2\cdots a_ja_{j+1}\cdots a_n$,*

* Research supported in part by NIH Grant RO1 GM62118 (to X.G.) and NSF of China (19771025). Wu is on leave from Math Dept,N.U.D.T.,Hunan 410073,China.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 86–90, 2001.
© Springer-Verlag Berlin Heidelberg 2001

$\mathbf{s}_2 = b_1 b_2 \cdots b_j b_{j+1} \cdots b_n \in \Sigma^n$, *a recombination for* $\mathbf{s}_1$ *and* $\mathbf{s}_2$ *at position* j $(1 \le j \le n-1)$ *is defined as the following operation:*

$$\begin{array}{rl} \text{From} & \mathbf{s}_1 = \underline{a_1 a_2 \cdots a_j}\ \underline{a_{j+1} \cdots a_n} \\ & \mathbf{s}_2 = \overline{b_1 b_2 \cdots b_j}\ \overline{b_{j+1} \cdots b_n} \\ \text{to} & \mathbf{t}_1 = \underline{a_1 a_2 \cdots a_j}\ b_{j+1} \cdots b_n \\ & \mathbf{t}_2 = \overline{b_1 b_2 \cdots b_j}\ \underline{a_{j+1} \cdots a_n}. \end{array}$$

It is denoted by $R(\mathbf{s}_1, \mathbf{s}_2; j; \mathbf{t}_1, \mathbf{t}_2)$, *or* $R(\mathbf{s}_1, \mathbf{s}_2; j)$. *Both* $\mathbf{s}_1$ *and* $\mathbf{s}_2$ *are called parents and both* $\mathbf{t}_1$ *and* $\mathbf{t}_2$ *are called children.*

Definition 2. *Let* $\mathcal{A}, \mathcal{S} \subseteq \Sigma^n$. $\mathcal{S}$ *is called a recombination ancestor of* $\mathcal{A}$ *if* $\mathcal{A}$ *can be generated from* $\mathcal{S}$ *by a series of recombinations, i.e., there exist a series of recombinations* $R_i(\mathbf{s}_1^{(i)}, \mathbf{s}_2^{(i)}; j^{(i)}; \mathbf{t}_1^{(i)}, \mathbf{t}_2^{(i)})$ $(i = 1, 2, \cdots, p)$ *such that (1)*$s_1^{(i)}, \mathbf{s}_2^{(i)} \in \mathcal{S} \cup \{\mathbf{t}_1^{(k)}, \mathbf{t}_2^{(k)} |\ 1 \le k \le i-1\}$, *(2)*$\mathcal{A} \subseteq \mathcal{S} \cup \{\mathbf{t}_1^{(i)}, \mathbf{t}_2^{(i)} |\ 1 \le i \le p\}$. $R_i(\mathbf{s}_1^{(i)}, \mathbf{s}_2^{(i)}; j^{(i)}; \mathbf{t}_1^{(i)}, \mathbf{t}_2^{(i)})$ $(i = 1, 2, \cdots, p)$ *is called a recombination process generating* $\mathcal{A}$ *from* $\mathcal{S}$. *It is called optimal if the number* p *of recombination steps is minimized among all such possible recombination processes.*

Recombination Problem 1. Let $\mathcal{A} \subseteq \Sigma^n$ and $\mathcal{S}$ a recombination ancestor of $\mathcal{A}$. Find the optimal recombination process generating $\mathcal{A}$ from $\mathcal{S}$. Denote $n(\mathcal{S}, \mathcal{A})$ the number of optimal recombination steps generating $\mathcal{A}$ from $\mathcal{S}$.

Example 1. Let $\mathcal{A} = \{\mathbf{a}_1 = 01010, \mathbf{a}_2 = 10001, \mathbf{a}_3 = 10110, \mathbf{a}_4 = 11001\}$ and $\mathcal{S} = \{\mathbf{s}_1 = 00101, \mathbf{s}_2 = 11010\}$. We have $n(\mathcal{S}, \mathcal{A}) = 4$ and the following is an optimal recombination process generating $\mathcal{A}$ from $\mathcal{S}$.

$$\begin{array}{rrrl} R_1(\mathbf{s}_1, \mathbf{s}_2;\ 1; \mathbf{a}_1, \mathbf{b}_4)\text{: From} & 0 & 0101 & = b_1 = \mathbf{s}_1 \\ & 1 & 1010 & = b_2 = \mathbf{s}_2 \\ \text{To} & 0 & 1010 & = b_3 = \mathbf{a}_1 \\ & 1 & 0101 & = b_4 \\ R_2(\mathbf{b}_4, \mathbf{s}_2; 2; \mathbf{b}_5, \mathbf{b}_6)\text{: From} & 10 & 101 & = b_4 \\ & 11 & 010 & = b_2 \\ \text{To} & 10 & 010 & = b_5 \\ & 11 & 101 & = b_6 \\ R_3(\mathbf{b}_4, \mathbf{s}_2; 3; \mathbf{a}_3,\ \mathbf{a}_4)\text{: From} & 101 & 01 & = b_4 \\ & 110 & 10 & = b_2 \\ \text{To} & 101 & 10 & = b_7 = \mathbf{a}_3 \\ & 110 & 01 & = b_8 = \mathbf{a}_4 \\ R_4(\mathbf{s}_1, \mathbf{b}_5; 3;\ \mathbf{a}_2, \mathbf{b}_9)\text{: From} & 001 & 01 & = b_1 \\ & 100 & 10 & = b_5 \\ \text{To} & 001 & 10 & = b_9 \\ & 100 & 01 & = b_{10} = \mathbf{a}_2 \end{array}$$

If $\mathcal{S}$ consists of only two sequences, then Problem 1 can be regarded as one with $|\Sigma| = 2$. For binary case, $\mathcal{S} = \{00\cdots0, 11\cdots1\}$ always generates an arbitrary binary sequence by recombinations. So it is a recombination ancestor for any

$\mathcal{A} \subseteq \{0,1\}^n$. Moreover, given any recombination ancestor $\mathcal{S} = \{s_{i1}s_{i2}\cdots s_{in} | i = 1,2\}$ of $\mathcal{A} = \{a_{i1}a_{i2}\cdots a_{in} | i = 1,2,\cdots,k\}$, for each $1 \leq j \leq n$, we define a function f_j on $\{s_{1j}, s_{2j}\}(s_{1j} \neq s_{2j})$ such that $f_j(s_{1j}) = 0$ and $f_j(s_{2j}) = 1$. Then we get $\mathcal{S}_0 = \{f_1(s_{11})f_2(s_{12})\cdots f_n(s_{1n}), f_1(s_{21})f_2(s_{22})\cdots f_n(s_{2n})\} = \{00\cdots0, 11\cdots1\}$ and $\mathcal{A}_0 = \{f_1(a_{i1})f_2(a_{i2})\cdots f_n(a_{in}) | i = 1,2,\cdots,k\}$. And $n(\mathcal{S},\mathcal{A}) = n(\mathcal{S}_0,\mathcal{A}_0)$. For $|\Sigma| = 2$, Recombination Problem 1 is reduced to the following simple form.

Recombination Problem 2. Let $\mathcal{A} \subseteq \{0,1\}^n$ be an arbitrary collection of binary sequences and $\mathcal{S} = \{00\cdots0, 11\cdots1\}$. Find the optimal recombination process generating $\mathcal{A}$ from $\mathcal{S}$.

In this paper, we discuss Problem 2 and find algorithms for it.

1.2 Terminology and Notation

Definition 3. *Let* $\mathbf{a} = a_1a_2\cdots a_p\cdots a_q\cdots a_n \in \{0,1\}^n$. $I = \mathbf{a}[p,q] = a_pa_{p+1}\cdots a_q$ *is called an alternative segment of* $\mathbf{a}$ *if* $a_i \neq a_{i+1}$ *for all* $p \leq i \leq q-1$. *An alternative block is a maximal alternative segment. Denote* $I_{\mathbf{a}} = \{I_1, I_2, \cdots, I_k\}$ *the collection of all alternative blocks of* $\mathbf{a}$. *Define the core of* $\mathbf{a}$ *as* $C_{\mathbf{a}} = a[s,t]$, *which is the minimum segment of* $\mathbf{a}$ *containing all* $I_i (1 \leq i \leq k)$. *Define the length of* $\mathbf{a}[p,q]$ *as* $l(\mathbf{a}[p,q]) = q-p$ *and the length of* $I_{\mathbf{a}}$ *as* $l(I_{\mathbf{a}}) = \sum_{j=1}^{k} l(I_j)$. *Denote* $\mathcal{P}(\mathcal{A}) = \{p | 1 \leq p \leq n-1, a_p \neq a_{p+1}$ *for some* $\mathbf{a} \in \mathcal{A}\}$. *For example, if* $\mathbf{a} = 11\underline{10101}1\underline{1010}000\underline{10}0$, *then* $I_1 = \mathbf{a}[3,7] = \underline{10101}$, $I_2 = \underline{1010}$, $I_3 = \underline{010}$. $I_{\mathbf{a}} = \{I_1, I_2, I_3\}$. $C_{\mathbf{a}} = \mathbf{a}[3,17] = \underline{10101}1\underline{101}000\underline{010}$. $l(I_1) = 4$ *and* $l(I_{\mathbf{a}}) = 9$.

Definition 4. *Let* $\mathbf{a}, \mathbf{b} \in \{0,1\}^n$ *with* $C_{\mathbf{a}} = \mathbf{a}[p,q]$ *and* $C_{\mathbf{b}} = \mathbf{b}[u,v]$. *If* $u \leq p \leq q \leq v$ *and* $\mathbf{a}[p,q] = \mathbf{b}[p,q]$, *we say that* $\mathbf{b}$ *covers* $\mathbf{a}$, *denoted by* $\mathbf{a} \subseteq \mathbf{b}$. $\mathbf{b}$ *and* $\mathbf{a}$ *are called disjoint or independent if* $q \leq u$, *or* $v \leq p$, *denoted by* $\mathbf{a} \cap \mathbf{b} = \emptyset$. *If* $q \leq u$, *we say* $\mathbf{a} < \mathbf{b}$. *For any alternative segments (or blocks)* I *and* J, *we similarly define* $I \subseteq J$, $I \cap J = \emptyset$, *and* $I < J$.

Definition 5. *Denote* $\mathcal{C}(\mathcal{A}) = \{C_{\mathbf{a}} | \mathbf{a} \in \mathcal{A}\}$. $\mathcal{A}$ *is called a nest of sequences if* $\mathbf{a} \subseteq \mathbf{b}$, *or* $\mathbf{b} \subseteq \mathbf{a}$ *for any* $\mathbf{a}, \mathbf{b} \in \mathcal{A}$. *It is called a tree of sequences if either* $\mathbf{a} \cap \mathbf{b} = \emptyset$, *or* $\mathbf{a} \subseteq \mathbf{b}$, *or* $\mathbf{b} \subseteq \mathbf{a}$ *for any* $\mathbf{a}, \mathbf{b} \in \mathcal{A}$. *For* $\mathbf{a}, \mathbf{b} \in \mathcal{A}$, *if* $\mathbf{b} \subseteq \mathbf{a}$ *and there exists no* $\mathbf{c} \in \mathcal{A}$ *such that* $\mathbf{b} \subseteq \mathbf{c} \subseteq \mathbf{a}$ *with* $\mathbf{c} \neq \mathbf{a}$ *and* $\mathbf{c} \neq \mathbf{b}$, *then* $\mathbf{b}$ *is called a branch of* $\mathbf{a}$. *Denote* $B_{\mathbf{a}}$ *the collection of all branches of* $\mathbf{a}$ *in* $\mathcal{A}$. *A branch is called a leaf (or youngest branch) if it has no branch in* $\mathcal{A}$. $\mathbf{a} \in \mathcal{A}$ *is called a root (or oldest branch) if it is not a branch of any other sequences in* $\mathcal{A}$.

1.3 Related Work

Our recombination problem is a generalization of the following problem(cf.[5]).

Recombination Distance. Given $\mathbf{a} = a_1a_2\cdots a_m$, $\mathbf{b} = b_1b_2\cdots b_n$, and $\mathbf{c} = c_1c_2\cdots c_k$, find the minimum cost recombination to produce $\mathbf{c}$ from $\mathbf{a}$ and $\mathbf{b}$.

The goal is to generate a third sequence from a given pair by recombinations consisting of multiple crossovers and point mutations. In our problems, we define a recombination consisting of a single crossover with no point mutations. We discuss a more general problem to construct the optimal recombination spanning history from one family of sequences to another one. Its general case with multiple crossovers and point mutations is NP-complete(cf.[5,6]).

2 Theorems and Algorithms

In this section, we show some theorems on optimal recombination processes and design a greedy algorithm for finding the optimal recombination process for a tree of binary sequences. We always assume $\mathcal{S} = \{00\cdots0,\ 11\cdots1\}$.

Theorem 1. *Let* $\mathbf{a} \in \{0,1\}^n$. *Then* $n(\mathcal{S}, \{\mathbf{a}\}) = l(I_{\mathbf{a}})$.

Proof. $\mathbf{a}$ is optimally generated by $R(\mathbf{s}_1, \mathbf{s}_2; p; \mathbf{b}_1, \mathbf{c}_1)$, $R(\mathbf{b}_1, \mathbf{c}_1; p+1; \mathbf{b}_2, \mathbf{c}_2)$, $R(\mathbf{b}_2, \mathbf{c}_2; p+2; \mathbf{b}_3, \mathbf{c}_3)$, $\cdots\ \cdots$, $R(\mathbf{b}_{q-p-1}, \mathbf{c}_{q-p-1}; q-1; \mathbf{b}_{q-p}, \mathbf{c}_{q-p}), \cdots$ over all positions in $\mathcal{P}(\{\mathbf{a}\})$. Therefore, $n(\mathcal{S}, \{\mathbf{a}\}) = l(I_{\mathbf{a}})$.

The series of recombinations in the proof is called a recombination extension and denoted by $ext(\mathbf{s_1}, \mathbf{a})$. Generally, if $\mathbf{a} \subseteq \mathbf{b}$ and $\mathbf{a}$ has been generated, then $\mathbf{b}$ can be generated by series of recombinations on the positions in $\mathcal{P}(\{\mathbf{b}\})$ but not $\mathcal{P}(\{\mathbf{a}\})$, called an recombination extension from $\mathbf{a}$ to $\mathbf{b}$ and denoted by $ext(\mathbf{a}, \mathbf{b})$.

Theorem 2. *If* $\mathcal{A}$ *is a nest, then* $n(\mathcal{S}, \mathcal{A}) = |\mathcal{P}(\mathcal{A})|$.

Proof. By Theorem 1 and recombination extensions.

Theorem 3. *Let* $\mathcal{A}_1, \mathcal{A}_2 \subseteq \{0,1\}^n$ *be independent nests, i.e.,* $\mathbf{a} \cap \mathbf{b} = \emptyset$ *for any* $\mathbf{a} \in \mathcal{A}_1, \mathbf{b} \in \mathcal{A}_2$. *Then* $n(\mathcal{S}, \mathcal{A}_1 \cup \mathcal{A}_2) = |\mathcal{P}(\mathcal{A}_1)| + |\mathcal{P}(\mathcal{A}_2)|$.

Proof. Apply Theorem 2 to $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively.

Theorem 4. *Let* $\mathcal{A}_1, \mathcal{A}_2 \subseteq \{0,1\}^n$ *be independent nests and* $\mathbf{a} \in \{0,1\}^n$. *For any* $\mathbf{b}_1 \in \mathcal{A}_1$ *and* $\mathbf{b}_2 \in \mathcal{A}_2$, *(1)*$\mathbf{b}_1 < \mathbf{b}_2$ *and (2)*$\mathbf{b}_1, \mathbf{b}_2 \subseteq \mathbf{a}$. *Then* $n(\mathcal{S}, \mathcal{A}_1 \cup \mathcal{A}_2 \cup \{\mathbf{a}\}) = |\mathcal{P}(\{\mathbf{a}\})| + 1$.

Proof. Choose a position p between $\mathcal{A}_1$ and $\mathcal{A}_2$, then $C_{\mathbf{a}}$ is partitioned into C_1 and C_2. Choose $\mathbf{a}_1$ and $\mathbf{a}_2$ with $C_{\mathbf{a}_1} = C_1$ and $C_{\mathbf{a}_2} = C_2$. Then the theorem follows by applying Theorem 2 to both $\mathcal{A}_1 + \{\mathbf{a}_1\}$ and $\mathcal{A}_2 + \{\mathbf{a}_2\}$ with one more recombination $R(\mathbf{a}_1, \mathbf{a}_2; p)$, i.e., $|\mathcal{P}(\{\mathbf{a}_1\})| + |\mathcal{P}(\{\mathbf{a}_2\})| + 1 = |\mathcal{P}(\{\mathbf{a}\})| + 1$ steps.

Theorem 5. *Let* $\mathcal{A}$ *be a tree. Then* $n(\mathcal{S}, \mathcal{A}) = |\mathcal{P}(\mathcal{A})| + \sum_{\mathbf{a} \in \mathcal{A}} (|B_{\mathbf{a}}| - 1)$.

Proof. By Theorem 4 and induction on $|\mathcal{A}|$. Choose a root $\mathbf{a} \in \mathcal{A}$ and denote $B_{\mathbf{a}} = \{\mathbf{d}_1 < \mathbf{d}_2 < \cdots < \mathbf{d}_p\}$. We define $ext(\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p; \mathbf{a})$ as the following recombination process: (1)Choose a position p_i between $\mathbf{d}_i$ and $\mathbf{d}_{i+1}$ $(1 \le i \le p-1)$, (2)Partition $C_{\mathbf{a}}$ by all p_i into C_i and choose $\mathbf{b}_i$ with $C_{\mathbf{b}_i} = C_i (1 \le i \le p)$, (3)Make $ext(\mathbf{d}_i, \mathbf{b}_i)$ to get $\mathbf{b}_i$ from $\mathbf{d}_i (1 \le i \le p)$, (4)Make the recombinations: $R(\mathbf{b}_1, \mathbf{b}_2; p_1; \mathbf{f}_1, \mathbf{g}_1)$ and $R(\mathbf{f}_{i-1}, \mathbf{b}_i; p_i; \mathbf{f}_i, \mathbf{g}_i)$ $(2 \le i \le p-1)$ with $\mathbf{b}_j \subseteq \mathbf{f}_i$ $(1 \le j \le i-1)$. Then $\mathbf{f}_{p-1} = \mathbf{a}$. $ext(\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p; \mathbf{a})$ generates $\mathbf{a}$ from $\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p$ in optimal steps. $\mathcal{A}$ is then generated from branches to roots by induction.

Based on Theorem 5, we now design a greedy algorithm for finding the optimal recombination spanning evolutionary history generating a tree $\mathcal{A}$ from $\mathcal{S}$.

Greedy Algorithm
Input: $\mathcal{A}$ (nest/tree).
Output: Optimal recombination process.
Step 1 Find $B_{\mathbf{a}}$ for all $\mathbf{a} \in \mathcal{A}$ and set $E = \emptyset$.
Step 2 While $\mathcal{A} - E \neq \emptyset$ {
Choose a leaf (youngest branch) $\mathbf{a} \in \mathcal{A} - E$ and
denote $B_{\mathbf{a}} = \{\mathbf{d}_1 < \mathbf{d}_2 < \cdots < \mathbf{d}_p\}$.
Generate $\mathbf{a}$ by $ext(\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p; \mathbf{a})$.
$E = E + \{\mathbf{a}\}$.
}

Theorem 6. *Greedy Algorithm finds the optimal recombination process for a tree $\mathcal{A}$ and the run time is $O(|\mathcal{A}|^2 n)$.*

Proof. For each pair $\mathbf{a}$ and $\mathbf{b}$ of sequences, the algorithm compares n positions to determine whether $\mathbf{a} \subseteq \mathbf{b}$ or not. There are $O(|\mathcal{A}|^2)$ pairs of sequences. Therefore the run time is $O(|\mathcal{A}|^2 n)$.

3 Conclusion

The greedy algorithm is designed to generate a tree from two sequences. The most interesting part of the problem is to generate an arbitrary collection of sequences from a given recombination ancestor. Our discussion may be applied to some problems in human SNP(single nucleotide polymorphism) genome project.

References

1. Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G., 1998. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids,*. Cambridge University Press.
2. Gusfield, D., 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press.
3. Hein, J., 1990. *Reconstruct evolution of sequences subject to recombination, Mathematical biosciences* 98:185-200.
4. Hein, J., 1993. *A heuristic method to reconstruct the history of sequences subject to recombination, Journal of Molecular Evolution* 36: 396-450.
5. Kececioglu, J. and Gusfield, D., 1994. *Reconstructing a history of recombinations from a set of sequences, 5th Annual ACM-SIAM Symposium on Discrete Algorithm* Arlington, Virginia, pp.471-480. Also in Discrete Applied Mathematics 88(1998)239-260.
6. Wang,L., B. Ma and M. Li, 2000, *Fixed topology alignment with recombination, Discrete Applied Mathematics* 104: 281-300

Generating Well-Shaped *d*-dimensional Delaunay Meshes

Xiang-Yang Li

Dept. of Computer Science, Illinois Institute of Technology,
10 W. 31st Street, Chicago, IL 60616.

Abstract. A d-dimensional simplicial mesh is a Delaunay triangulation if the circumsphere of each of its simplices does not contain any vertices inside. A mesh is well-shaped if the maximum aspect ratio of all its simplices is bounded from above by a constant. It is a long-term open problem to generate well-shaped d-dimensional Delaunay meshes for a given polyhedral domain. In this paper, we present a refinement-based method that generates well-shaped d-dimensional Delaunay meshes for any PLC domain with no small input angles. Furthermore, we show that the generated well-shaped mesh has $O(n)$ d-simplices, where n is the smallest number of d-simplices of any almost-good meshes for the same domain. A mesh is almost-good if each of its simplices has a bounded circumradius to the shortest edge length ratio.
Keywords: Mesh generation, Delaunay triangulation, well-shaped, aspect ratio, radius-edge ratio, computational geometry, algorithms.

1 Introduction

Mesh generation is the process of breaking a geometry domain into a collection of primitive elements. In this paper we exclusively consider d-dimensional simplicial Delaunay meshes. The *aspect-ratio* of a mesh is the maximum aspect-ratio among all of its simplicial elements. A mesh is *well-shaped* if its aspect ratio is bounded from above by a small constant. The *aspect ratio* of a simplex is usually defined as the ratio of its circumradius to its inradius. Generating well-shaped d-dimensional Delaunay meshes is one of the long-term open problems in mesh generation when $d > 2$. An alternative but weaker quality measurement is to use the *radius-edge ratio* [8]. It is the circumradius divided by the shortest edge length of the simplex. The radius-edge ratio of a mesh is the maximum radius-edge ratio among all of its elements. A mesh is *almost-good* if it has a small radius-edge ratio. Numerous methods [3,8,9,10] guarantee to generate almost-good 3-dimensional Delaunay meshes.

Bern *et al.* [1] showed that any set of n d-dimensional points had a Steiner Delaunay triangulation with $O(n^{\lceil d/2 \rceil})$ simplices, none of which has an obtuse dihedral angle. No bound depending only on n is possible if we require a bound on the minimum dihedral angle [1]. We assume a d-dimensional piecewise linear complex (PLC) domain Ω as the input. Shewchuk [11] showed that if each

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 91–100, 2001.
© Springer-Verlag Berlin Heidelberg 2001

k-dimensional constraining facet in Ω with $k \leq d-2$ is a union of strongly Delaunay k-simplices, then Ω has a d-dimensional constrained Delaunay triangulation. The Delaunay refinement method [3,9,10] can be extended to d-dimensions to generate almost-good Delaunay meshes if Ω satisfying the projection lemma [10] and having small input angles. However, it encounters significant difficulties in producing well-shaped Delaunay meshes in 3D. Chew [4] first proposed to add a vertex inside the circumsphere of any given badly-shaped tetrahedron (sliver) to remove it. The difficulty is in proving the existence of, and finding, a Steiner vertex that does not itself result in slivers. To make it possible, Chew defined a tetrahedron to be a sliver if it has a large aspect ratio, a small radius-edge ratio and its circumradius is no more than one unit. Without the restriction on the circumradius length, almost all results in [4] donot hold any more. In addition, with this restriction, the termination of his algorithm is straightforward. Recently, Li [6] extended this algorithm and showed how to generate well-shaped non-uniform Delaunay meshes in 3D.

In this paper, we show how to generate d-dimensional well-shaped meshes for a PLC domain with no small angles. In Section 2, we review the basic concepts and define what is a d-dimensional sliver simplex. [1] Then we give an algorithm in Section 3 to generate d-dimensional well-shaped Delaunay meshes. It basically adds a point around the circumcenter of each of d-simplices containing any k-dimensional sliver (hereafter, k-sliver). We prove its correctness in Section 4 and its termination guarantee in Section 5 by showing that the distance between the closest mesh vertices is just decreased by a constant factor compared with that of the input mesh. In Section 6, we show that the size of the generated mesh is within a constant factor of the size of any almost-good mesh generated for the same domain. Section 7 concludes the paper with discussions.

2 Preliminaries

After inserting a new vertex p, every new d-dimensional simplex created in the Delaunay triangulation of the new vertex set has p as one of its vertices. The new triangulation can be updated by efficient operations local to the vertex p. A sphere centered at a point c with a radius r is denoted as (c, r) hereafter. It is called *empty* if it does not contain any mesh vertices inside. The nearest neighbor graph defined by a d-dimensional vertex set is contained in the Delaunay triangulation of the vertex set. Thus the shortest edge length of the Delaunay triangulation is the closest distance among mesh vertices. This fact is used in proving the termination guarantee of our algorithm.

Delaunay refinement methods have been shown to be effective in generating almost-good meshes in 2 and 3 dimensions [9,10]. There is also no much difficulty to extend them to d-dimensions, if the input domain satisfies a projection lemma [10], to generate meshes with radius-edge ratio no more than $\sqrt{2}^{d-1}$.

[1] Surprisingly, a simple extension of 3-dimensional sliver definition to d-dimensions does not work here.

For a k-simplex μ, its *min-circumsphere* is the smallest d-dimensional sphere containing the vertices of μ on its surface. A point *encroaches* the domain boundary if it is contained inside the min-circumsphere of a boundary k-simplex μ. Here a k-simplex μ is a boundary simplex if it belongs to the Delaunay triangulation of a k-dimensional input boundary polyhedral face. A k_1-simplex τ_1 *directly* encroaches another k_2-simplex τ_2 if the circumcenter of τ_1 encroaches τ_2. A k_1-simplex τ_1 *indirectly* encroaches another k_2-simplex τ_2 if the circumcenter of τ_1 encroaches a simplex μ and μ directly or indirectly encroaches τ_2. Assume the circumcenter of a k_1-simplex τ encroaches a boundary k_2-simplex μ. Call τ the *encroaching simplex* and μ the *encroached simplex*. Assume that μ contains the projection of the circumcenter of τ inside. Then Shewchuk [10] showed that the circumradii of τ and μ satisfy that $R_\mu \geq \frac{1}{\sqrt{2}} R_\tau$.

A d-simplex is *bad* if it has a large aspect ratio. Let's consider a k-simplex τ, where $1 \leq k \leq d$. For later convenience, we use R_τ, L_τ and $\rho(\tau) = R_\tau / L_\tau$ to denote the circumradius, the shortest edge length and the radius-edge ratio of τ. Let V be its volume. We define $\sigma = \sigma(\tau) = V / L_\tau^k$ as a measure of its quality. Let ϱ_0 and σ_k, $1 \leq k \leq d$ be positive constants that we specify later.

Definition 1. *A k-simplex τ is* well-shaped *if $\rho(\tau) \leq \varrho_0$ and $\sigma(\tau) \geq \sigma_k$.*

Definition 2. [Sliver] *Call a k-simplex τ a* k-sliver *if $\rho(\tau) \leq \varrho_0$, $\sigma(\tau) < \sigma_k$, and each of its facets is well-shaped if $k > 3$.*

We call a d-simplex τ a *sliver-simplex* if it contains a k-sliver. If a d-simplex τ has a small radius-edge ratio and a small σ value, then it must have a face χ that is a sliver or τ itself is a d-sliver. It is not difficulty to prove that the volume of a d-simplex τ is at most $\varphi_d R_\tau^d$, where $\varphi_d = 2 \prod_{i=2}^{d} \frac{(i-1)^{(i-1)/2}(i+1)^{(i+1)/2}}{i^{i+1}}$. The aspect ratio of τ is then at most $\frac{(1+d)\varphi_{d-1}\rho(\tau)^d}{d\sigma(\tau)}$. This verifies our definition of σ.

3 Refinement Algorithm

For a k-simplex τ, we call the intersection of its min-circumsphere with the affine space defined by its vertices as its k-sphere. When refine a k-simplex τ, we add a point p inside the shrinked k-sphere $(c_\tau, \delta R_\tau)$, where $\delta < 1$ is a constant. The point p is *good* if its insertion will not introduce any small slivers in the new Delaunay triangulation. Here a created sliver μ is *small* if $R_\mu \leq b R_\tau$ for a constant b specified later. We call the solid k-dimensional ball $(c_\tau, \delta R_\tau)$ the *picking region* of τ, denoted by $\mathcal{P}(\tau)$. Its volume is $\phi_k \delta^k R_\tau^k$, where $\phi_k = \frac{2\pi^{k/2}}{k\Gamma(\frac{k}{2})}$.

Algorithm: Refinement (ϱ_0, σ_0, δ, b)

Enforce Empty Encroachment: Add the circumcenter c_τ of any encroached boundary simplex τ and update the Delaunay triangulation. If c_τ encroaches any lower dimensional boundary simplex μ, add c_μ instead of c_τ.

Clean Large Radius-Edge Ratio: Add the circumcenter c_τ of any d-simplex τ with a large $\rho(\tau)$ and update the Delaunay triangulation. If c_τ encroaches any boundary k-simplex, we apply the last rule instead of adding c_τ.

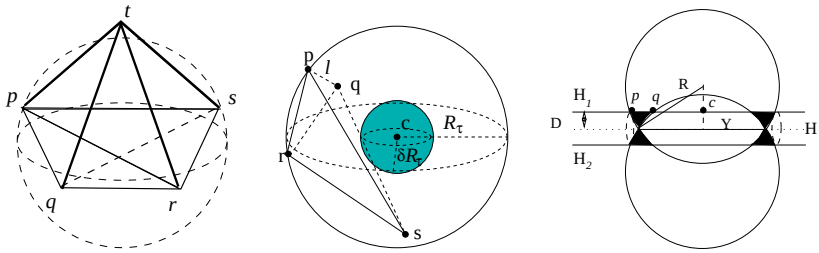


Fig. 1. Left: a 4-dimensional sliver example; Middle: the picking region of a simplex; Right: the forbidden region of μ with circumradius Y. Here $R = 2\varrho_0 Y$ and $D = 2k\sigma_0 Y$.

Clean the Slivers: For a sliver-simplex τ, add a good point $p \in \mathcal{P}(\tau)$ and update the Delaunay triangulation. If the circumcenter c_τ encroaches the domain boundary, we apply the following rules instead of finding p. If the insertion of p introduces some new d-simplices with large radius-edge ratio, we apply the previous rule to eliminate them immediately.

Encroach Boundary Simplices: If a boundary k-simplex μ is encroached directly or indirectly by a d-simplex with a large radius-edge ratio, add the circumcenter c_μ and update the Delaunay triangulation. However, if c_μ encroaches any other lower dimensional boundary simplex μ_1, we insert the circumcenter of μ_1 instead of adding c_μ.

If a boundary k-simplex μ is encroached directly or indirectly by a sliver-simplex, add a good point $p \in \mathcal{P}(\mu)$ and update the Delaunay triangulation. However, if c_μ encroaches any other lower dimensional boundary simplex μ_1, we add a good point from $\mathcal{P}(\mu_1)$ instead of adding p.

The key part of the algorithm is to find a good point p to refine a sliver-simplex τ. We select some k random points from $\mathcal{P}(\tau)$. Then choose the point that optimizes the quality σ of all created small simplices with a radius-edge ratio less than ϱ_0. Another approach is based on a randomized selection [4,6,7]. We randomly select a point $p \in \mathcal{P}(\tau)$ until we find a good point p. By defining sliver and small slivers properly, we can show that we are expected to find a good point p in constant rounds. Then the rest of the paper is devoted to prove the termination guarantee and the good grading guarantee of the algorithm.

4 Proofs of Correctness

4.1 Sliver Regions

Recall that a k-simplex τ is a sliver if $\rho(\tau) \leq \varrho_0$, $\sigma(\tau) \leq \sigma_k$, and all of its facets are well-shaped. The quality measure $\sigma(\tau)$ is related with a distance-radius ratio defined in [2,5]. Consider any vertex p of τ. Let μ be the facet formed by other vertices of τ. Let D be the Euclidean distance of point p from the hyperplane passing through μ. Recall R_μ is the circumradius of μ.

Lemma 1. *For any k-simplex τ, $\frac{D}{R_\mu} \leq 2k\frac{\sigma(\tau)}{\sigma(\mu)}$.*

PROOF. It follows from the volume of τ: $V = \sigma(\tau)L_\tau^k = \frac{D}{k}\sigma(\mu)L_\mu^{k-1}$. □

A k-simplex τ is a sliver if $\rho(\tau) \leq \varrho_0$ and $\sigma(\tau) = \frac{V}{L_\tau^k} \leq \sigma_0^k$ for a small constant σ_0, i.e., $\sigma_k = \sigma_0^k$. Consequently, we have $\frac{D}{R_\mu} \leq 2k\sigma_0$ if τ is a k-sliver.

Given a well-shaped k-simplex τ, a point p that forms a $(k+1)$-sliver together with τ can not be anywhere. It must be around the circumsphere of τ and is not far-away from the hyperplane passing τ. We call the locus of such point p as the *sliver region* or *forbidden region* F_τ of the simplex τ. It is easy to show that F_τ is contained in the solid sun-hour glass shaped region as illustrated in the right figure of Figure 1. Let $|F_\tau|$ denote the volume of F_τ.

Lemma 2. *For any well-shaped k-simplex τ, $|F_\tau| \leq c_d\sigma_0^{d-k+1}R_\tau^d$, where c_d is a constant depending only on ϱ_0, k, and d.*

PROOF. We know that F_τ is inside the slab region defined by the two hyperplanes H_1 and H_2 illustrated by the right figure of Figure 1. Let r_2, r_1 be the circumradius of the two spheres intersected by H_1 and the circumsphere of τ. Here $r_2 = ||cp||$ and $r_1 = ||cq||$. It is easy to show that $r_2^2 \leq (1-(2k\sigma_0)^2 + 8k\sigma_0\varrho_0)\cdot Y^2$ and $r_1^2 \geq (1-(2k\sigma_0)^2 - 8k\sigma_0\varrho_0)\cdot Y^2$. The volume of a k-sphere with the radius r is $\phi_k r^k$. Then

$$|F_\tau| \leq \phi_k(r_2^k - r_1^k)\cdot\phi_{d-k}D^{d-k}.$$

Notice $\forall i \geq 1$, $r_2^{i+2} - r_1^{i+2} \leq (r_2^2 + r_1^2)(r_2^i - r_1^i)$. In addition, $r_2^2 + r_1^2 = 2(1-(2k\sigma_0)^2)Y^2 \leq 2Y^2$, and $r_2^2 - r_1^2 \leq 16k\varrho_0\sigma_0Y^2$. The fact that $r_2 \geq Y$ implies that $r_2 - r_1 \leq 16k\varrho_0\sigma_0 Y$. By induction, we have $r_2^k - r_1^k \leq 2^{\lceil k/2\rceil+3}k\varrho_0\sigma_0Y^k$. Consequently, we have $|F_\tau| \leq \phi_k\phi_{d-k}2^{\lceil k/2\rceil+3}k\varrho_0\sigma_0(2k\sigma_0)^{d-k}Y^d$. The lemma follows by setting $c_d = \phi_k\phi_{d-k}2^d k^{d-k+1}\varrho_0 > \phi_k\phi_{d-k}2^{\lceil k/2\rceil+3}k\varrho_0(2k)^{d-k}$. □

A good point p used to refine a k-simplex τ cannot be inside $F_\mu \cap \mathcal{P}(\tau)$ for any well-shaped m-simplex μ, We use $F_{\mu,k}$ to denote such intersection region.

Lemma 3. *For any k-simplex τ and a well-shaped m-simplex μ, $|F_{\mu,k}| \leq c_k\sigma_0 R_\mu^k$, where c_k is a constant that depends only on ϱ_0, d and k.*

4.2 Existence

We then show that given a k-simplex τ in an almost-good d-dimensional simplicial Delaunay mesh, there is a good point p in $\mathcal{P}(\tau)$. Let $S(\tau)$ be the set of simplices in any dimensions that each can form a small Delaunay sliver with a point from $\mathcal{P}(\tau)$. A created sliver μ is Delaunay if it belongs to the Delaunay triangulation by inserting a point from $\mathcal{P}(\tau)$; it is small if $R_\mu \leq bR_\tau$.

We first recall some results from Talmor *et al.* and extend it to d-dimensions. For a mesh vertex v, the *edge length variation*, denoted by $\nu(v)$, is defined as the length of the longest edge incident on it divided by the length of the shortest edge incident on it. Talmor[12] proved that, given an almost-good mesh in 3D, the edge length variation of each mesh vertex is at most a constant depending on the radius-edge ratio of the mesh. This result is extended to d-dimensions.

Lemma 4. *Given any vertex v of a d-dimensional almost good mesh, $\nu(v) \leq \nu_0$, where ν_0 is a constant depending on the mesh's radius-edge ratio ϱ_0.*

Lemma 5. *Given a k-simplex τ of an almost-good mesh, the number of the vertices of $S(\tau)$ is at most a constant depending on ν_0 and b.*

PROOF. Assume that a well-shaped m-simplex μ forms a Delaunay sliver together with a point $p \in \mathcal{P}(\tau)$. The edges of μ have lengths at most $2R_\mu \leq 2bR_\tau$. The edges incident on any vertex q of μ before point p is introduced have length at least $2bR_\tau/\nu_0$. It implies that the closest distance among all vertices from $S(\tau)$ is at least $2bR_\tau/\nu_0$. It is simple to show that all such vertices are inside the sphere centered at c_τ with a radius $(\delta + 2b)R_\tau$. Then by a volume argument, the number of vertices of $S(\tau)$ is a constant. □

Thus the number of simplices in $S(\tau)$ is also a constant, let's say, W.

Theorem 1. *Good point exists in any k-simplex τ of an almost-good mesh.*

PROOF. Each simplex μ from $S(\tau)$ claims a forbidden region F_μ with volume at most $c_k \cdot \sigma_0 R_\mu^k$. The circumradius R_μ of μ is at most bR_τ. The volume of $\mathcal{P}(\tau)$ is $\phi_k \delta^k R_\tau^k$. Therefore, if we select σ_0 such that $W \cdot c_k \sigma_0 b^k R_\tau^k < \phi_k \delta^k R_\tau^k$, then the pigeonhole principal will guarantee the existence of a good point $p \in \mathcal{P}(\tau)$. □

5 Termination Guarantee

A k-sliver is called *created* if it contains at least one Steiner vertex; otherwise it is called *original*. We classify the bad d-simplices into three categories. The simplices containing an *original k-sliver* are called *original sliver-simplices*. The simplices containing only *created k-slivers* are called *created sliver-simplices*. The third category has d-simplices with a large radius-edge ratio. Let $e(\tau)$ be the shortest edge length introduced by eliminating a d-simplex τ. Here $e(\tau)$ could be less than L_τ for a sliver-simplex τ. For simplicity, we assume that all original sliver-simplices are removed first.

Lemma 6. *For an original sliver-simplex μ, $e(\mu) \geq \frac{(1-\delta)}{\sqrt{2}^{d+1}} L_\tau$.*

Lemma 7. *For a simplex τ with $\frac{R_\tau}{L_\tau} > \varrho_0$, $e(\tau) \geq \frac{\varrho_0}{\sqrt{2}^{d-1}} L_\tau$.*

The proofs are similar to the 3-dimensional counterpart, which are omitted. It remains to show that refining any created sliver-simplex will not introduce shorter edges to the mesh. Let's consider a sliver-simplex μ. Assume it contains a k-sliver τ created by inserting a point from the picking region of an element, say $f(\tau)$. Element $f(\tau)$ is called the *parent* of the k-sliver τ. There are three cases: $f(\tau)$ is a sliver-simplex; $f(\tau)$ is a d-simplex with a large radius-edge ratio; $f(\tau)$ is an encroached boundary simplex.

Lemma 8. *Assume a sliver-simplex μ contains a k-sliver τ created by splitting another sliver-simplex $f(\tau)$. Then $e(\mu) \geq \frac{(1-\delta)b}{\sqrt{2}^{d+1}} \cdot L_{f(\tau)}$.*

Lemma 9. *Assume a sliver-simplex μ contains a k-sliver τ, and parent $f(\tau)$ has a large radius-edge ratio. Then $e(\mu) \geq \frac{(1-\delta)\varrho_0}{\sqrt{2}^{d+1}} \cdot L_{f(\tau)}$.*

Lemma 10. *Assume a sliver-simplex μ contains a sliver τ created by splitting a boundary m-simplex χ that is encroached directly or indirectly by $p(\chi)$. Then*

1. *$e(\mu) \geq \frac{(1-\delta)\varrho_0 L_{p(\chi)}}{2^d}$, if d-simplex $p(\chi)$ has a large radius-edge ratio.*
2. *$e(\mu) \geq \frac{(1-\delta)bL_\chi}{\sqrt{2}^{d+1}}$, if d-simplex $p(\chi)$ is a sliver-simplex.*

Combining all above analysis, we know that the shortest distance between all mesh vertices is at least $\frac{1-\delta}{\sqrt{2}^{d+1}}$ factor of that of the original mesh if we choose b, δ and ϱ_0 such that $(1-\delta)b \geq \sqrt{2}^{d+1}$ and $(1-\delta)\varrho_0 \geq 2^d$. We then have the following theorem by a volume argument.

Theorem 2. *The algorithm terminates in generating well-shaped Delaunay meshes.*

6 Good Grading Guarantee

As [6,7,10], we study the relation between the nearest neighbor function defined by the final mesh and the local feature size function defined by the input domain. The local feature size $lfs(x)$ of a point x is the radius of the smallest ball intersecting two non-incident segments or vertices. The nearest neighbor $N(x)$ of a point x is its distance to the second nearest mesh vertex.

With each vertex v, we associate an *insertion edge length* e_v equal to the length of the shortest edge connected to v immediately after v is introduced into the Delaunay mesh. Notice that v may not have to be inserted into the mesh actually. If v is an input vertex, then e_v is the distance between v and its nearest input neighbor. So $e_v \geq lfs(v)$ from the definition of $lfs(v)$. If v is from the picking region of a sliver-simplex τ, then $(1+\delta)R_\tau \geq e_v \geq (1-\delta)R_\tau$. The inequality $(1+\delta)R_\tau \geq e_v$ comes from the fact that there is a mesh vertex on the circumsphere of τ.

For the sake of convenience of analyzing, we also define a parent vertex $p(v)$ for each vertex v, unless v is an input vertex. Intuitively, for any noninput vertex v, $p(v)$ is the vertex "responsible" for the insertion of v. We discuss in detail what means by responsible here. Assume v is inserted inside the picking region of a simplex μ. If simplex μ has $\rho(\mu) \geq \varrho_0$, then $p(v)$ is the most recently inserted end point of the shortest edge of μ. If μ is a sliver-simplex containing an original k-sliver τ, then $p(v)$ is an end point of the shortest edge of τ. If μ is a created sliver-simplex containing a created k-sliver τ, then $p(v)$ is the most recently inserted vertex of τ. If μ is an encroached boundary k-simplex, then $p(v)$ is the encroaching vertex. For the sake of simplicity, we always use the almost-good

mesh generated by Delaunay refinement method as input mesh. Therefore the boundary simplices can not be encroached by input vertices. Ruppert [9] and Shewchuk [10] showed that $N()$ defined on the mesh generated by Delaunay refinement method is within a constant factor of $lfs(v)$, i.e., $N(v) \sim lfs(v)$.

Lemma 11. *Let v be a vertex in the final mesh and let $p = p(v)$. Then we have $e_v \geq lfs(v)$ for an input vertex v; and $e_v \geq C \cdot e_p$ for a Steiner point v, where C is a constant specified in the proof.*

PROOF. If v is an original input vertex, then $e_v \geq lfs(v)$ from the definition of $lfs(v)$. Thus the theorem holds. Then consider a non-input vertex v. We first consider that v is selected from the picking region of a d-simplex; say μ.

Case 1.1: μ is a d-simplex with large radius-edge ratio $\rho(\mu) \geq \varrho_0$. The parent p is one of the end points of the shortest edge of μ. Let L_μ be the length of the shortest edge pq of μ. Then q is an original vertex or is inserted before p. Therefore, $e_p \leq \|p - q\| = L_\mu \leq \frac{R_\mu}{\varrho_0}$. Thus $e_v = R_\mu \geq \varrho_0 \cdot e_p$.

Case 1.2: μ is a sliver-simplex containing an original k-sliver τ. The parent $p = p(v)$ is one of the end points of the shortest edge of τ. Let L_τ be the length of the shortest edge pq of τ. Similar to the previous case, we have $e_p \leq \|p - q\| = L_\tau$. Notice that $R_\mu \geq R_\tau \geq L_\tau/2$. Thus $e_v \geq (1 - \delta)R_\mu \geq \frac{1-\delta}{2} \cdot e_p$.

Case 1.3: μ is a sliver-simplex containing a created k-sliver τ. There are three cases about the parent element $f(\tau)$ of τ: $f(\tau)$ is a sliver; $f(\tau)$ has a large radius-edge ratio; $f(\tau)$ is an encroached k-dimensional boundary simplex. Recall that the parent vertex $p = p(v)$ is the most recently inserted vertex of τ.

Subcase 1.3.1: $f(\tau)$ is a sliver. Recall that the insertion of p from the picking region of sliver $f(\tau)$ will always avoid creating small slivers. Thus, $R_\tau \geq bR_{f(\tau)}$, where $R_{f(\tau)}$ is the circumradius of $f(\tau)$. Notice that $e_p \leq (1 + \delta)R_{f(\tau)}$. Thus, $e_v \geq (1 - \delta)bR_{f(\tau)} \geq \frac{1-\delta}{1+\delta} b \cdot e_p$.

Subcase 1.3.2: $f(\tau)$ has $\rho(f(\tau)) \geq \varrho_0$. Let pq be an edge of simplex τ, where p is inserted from the picking region of $f(\tau)$. Notice that $e_p \leq \|p - q\|$. We also have $R_\tau \geq \|p - q\|/2$. Thus, $e_v \geq (1 - \delta)R_\mu \geq (1 - \delta)R_\tau \geq \frac{1-\delta}{2} \cdot e_p$.

Final subcase 1.3.3: $f(\tau)$ is an encroached boundary k-simplex. We first consider the scenario that $f(\tau)$ is encroached by a sliver-simplex directly or indirectly. We then know that the insertion of p from $f(\tau)$ will always avoid creating small slivers because the mesh is almost-good. Thus $R_\tau \geq bR_{f(\tau)}$. Notice that $e_p \leq (1 + \delta)R_{f(\tau)}$. Thus, $e_v \geq (1 - \delta)bR_{f(\tau)} \geq \frac{1-\delta}{1+\delta} b \cdot e_p$. We then consider the scenario that $f(\tau)$ is encroached by a d-simplex with a large radius-edge ratio directly or indirectly. Here parent p is selected from the picking region of $f(\tau)$. Let pq be an edge of τ. Notice that $e_p \leq \|p - q\|$. We also have $R_\tau \geq \|p - q\|/2$. Thus, $e_v \geq (1 - \delta)R_\mu \geq (1 - \delta)R_\tau \geq \frac{1-\delta}{2} \cdot e_p$.

Then we consider that v is selected from the picking region of a boundary k-simplex χ, which is encroached by a m-simplex μ.

Case 2.1: μ is encroached by a sliver-simplex directly or indirectly. Here parent p is always the circumcenter of μ. Notice that $R_\mu \leq \sqrt{2}R_\chi$ and $e_p \leq (1 + \delta)R_\mu$. Thus, $e_v \geq (1 - \delta)R_\chi \geq \frac{1-\delta}{\sqrt{2}(1+\delta)} \cdot e_p$.

Case 2.2: μ is encroached directly or indirectly by a d-simplex with a large radius-edge ratio. Parent p is the circumcenter of μ. We always have $R_\mu \leq \sqrt{2}R_\chi$ and $e_p = R_\mu$. Thus, $e_v = R_\chi \geq R_\mu\sqrt{2} \geq \frac{\sqrt{2}}{2} \cdot e_p$. □

For a vertex v, as [10], we define $D_v = \frac{lfs(v)}{e_v}$. We call D_v the *density ratio* at point v. Clearly, initially D_v is at most one for an input vertex v, and after inserting new vertices, D_v tends to become larger.

Lemma 12. *Let v be a vertex with a parent $p = p(v)$ if there is any. Assume that $e_v \geq C \cdot e_p$. If v is inserted due to eliminating sliver-simplex, then $D_v \leq \frac{1+\delta}{1-\delta} + \frac{D_p}{C}$. If v is inserted beause of eliminating a d-simplex with a large radius-edge ratio, then $D_v \leq 1 + \frac{D_p}{C}$.*

The proof is omitted, which is almost the same as the 3-dimensional counterpart.

Theorem 3. *There are fixed constants $D_k \geq 1$, $1 \leq k \leq d$ such at for any vertex v inserted or rejected at the picking region of a bad k-simplex, $D_v \leq D_k$. Specifically, the values of D_i should satisfy the following conditions:*

$$D_d \geq max\{\frac{\varrho_0 A + \sqrt{2}^{d+1} B_{d-2}}{\varrho_0(1-\delta) - 2^d\alpha^{d-1}}, \frac{\alpha b + \alpha^2 + \alpha B_{d-2}}{b - \sqrt{2}^{d-1}\alpha^d}, \alpha + \frac{2}{1-\delta}\},$$

$$D_{d-k} = B_{k-1} + \sqrt{2}^k \alpha^k D_d,$$

where $1 \leq k \leq d-1$, $A = \alpha(1-\delta) + \frac{\sqrt{2}^{d+1}-2}{\sqrt{2}-1}$, and $B_i = \sum_{j=1}^{i}(\sqrt{2}^j \alpha^{j+1})$. Hence, there is a constant $D = \max_{k=1}^{d}\{D_k\}$ such that $D_v \leq D$ for all mesh vertex v.

The proof, which is based on induction, of the theorem is omitted due to space limit. The following theorem concludes that the generated mesh has good grading, i.e., for any mesh vertex v, $N(v)$ is at least some constant factor of $lfs(v)$.

Theorem 4. *For any mesh vertex v generated by refinement algorithm, the distance connected to its nearest neighbor vertex u is at least $\frac{lfs(v)}{D+1}$.*

The proof is omitted here, which is the same as the three-dimensional counterpart. Thus, if $\varrho_0 > 2^d\alpha^d$ and $b > \sqrt{2}^{d-1}\alpha^d$, our algorithm generates well-shaped Delaunay meshes with good grading. Ruppert showed that the nearest neighbor value $N(v)$ of a mesh vertex v of any almost-good mesh is at most a constant factor of $lfs(v)$, where the constant depends on the radius-edge ratio of the mesh. The above Lemma 4 shows that the nearest neighbor $N(v)$ for the well-shaped Delaunay mesh is at least some constant factor of $lfs(v)$. Notice that the number of vertices of an almost-good mesh is $O(\int_{x\in\Omega} \frac{1}{N(x)^d} dx)$. Then we have the following theorem.

Theorem 5. *Given a d-dimensional almost-good mesh with n vertices, the generated well-shaped mesh has $O(n)$ vertices, where the constant depends on d and the radius-edge ratios of the meshes.*

7 Discussions

In this paper, we present a refinement-based method that guarantees to remove all slivers in a d-dimensional almost-good simplicial mesh. Notice that the σ_0 derived from all the proofs may be too small for any practical use. We would like to conduct some experiments to see what σ_0 can guarantee that there is no small slivers created. In addition, we could have different definitions about slivers depending upon the location of the simplex: inside or near the domain boundary. This could also improve the bound on σ_0. Let ρ_d be the minimum radius-edge ratio of a d-simplex. It is easy to show that $\rho_2 = \frac{\sqrt{3}}{3}$ and $\rho_d = \frac{1}{2\sqrt{1-\rho_{d-1}^2}}$, which is much less than the radius-edge ratio bound ϱ_0 achieved by our algorithm (and also the Delaunay refinement). We would like to know if we can get better radius-edge ratio bound on the d-simplices of the generated mesh.

References

1. Bern, M., Chew, P., Eppstein, D., and Ruppert, J. Dihedral bounds for mesh generation in high dimensions. In *Proc. 6th ACM-SIAM Symposium on Discrete Algorithms* (1995), ACM, pp. 189–196.
2. Cheng, S. W., Dey, T. K., Edelsbrunner, H., Facello, M. A., and Teng, S. H. Sliver exudation. In *Proc. 15th ACM Symposium on Computational Geometry* (1999), pp. 1–13.
3. Chew, L. P. Constrained Delaunay triangulations. *Algorithmica 4* (1989), 97–108.
4. Chew, L. P. Guaranteed-quality delaunay meshing in 3d (short version). In *13th ACM Sym. on Comp. Geometry* (1997), pp. 391–393.
5. Edelsbrunner, H., Li, X. Y., Miller, G., Stathopoulos, A., Talmor, D., Teng, S. H., ÜNGÖR", A., and Walkington, N. Smoothing and cleaning up slivers. In *ACM Symposium on Theory of Computing (STOC00)* (2000).
6. Li, X. Y. *Sliver-free Three Dimensional Delaunay Mesh Generation.* PhD thesis, University of Illinois at Urbana-Champaign, 2000.
7. Li, X. Y., and Teng, S. H. Sliver-free three dimensional delaunay mesh generation. In *Symposium on Discrete Algorithms (SODA)* (2001).
8. Miller, G. L., Talmor, D., Teng, S. H., and Walkington, N. A delaunay based numerical method for three dimensions: generation, formulation, and partition. In *Proc.* 27^{th} *Annu. ACM Sympos. Theory Comput.* (1995), pp. 683–692.
9. Ruppert, J. A new and simple algorithm for quality 2-dimensional mesh generation. In *Third Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 83–92.
10. Shewchuk, J. R. *Delaunay Refinement Mesh Generation.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1997.
11. Shewchuk, J. R. A condition guaranteeing the existence of higher-dimensional constrained delaunay triangulations. In *Proc. 14th Annual Symposium on Computational Geometry* (1998), ACM, pp. 76–85.
12. Talmor, D. *Well-Spaced Points for Numerical Methods.* PhD thesis, Carnegie Mellon University, 1997.

Towards Compatible Triangulations

Oswin Aichholzer[1,⋆], Franz Aurenhammer[1],
Hannes Krasser[1,⋆⋆], and Ferran Hurtado[2,⋆⋆⋆]

[1] Institute for Theoretical Computer Science
Graz University of Technology, Graz, Austria
{oaich,auren,hkrasser}@igi.tu-graz.ac.at
[2] Departament de Matematica Aplicada II
Universitat Politecnica de Catalunya, Barcelona, Spain
hurtado@ma2.upc.es

Abstract. We state the following conjecture: any two planar n-point sets (that agree on the number of convex hull points) can be triangulated in a compatible manner, i.e., such that the resulting two planar graphs are isomorphic. The conjecture is proved true for point sets with at most three interior points. We further exhibit a class of point sets which can be triangulated compatibly with any other set (that satisfies the obvious size and hull restrictions). Finally, we prove that adding a small number of Steiner points (the number of interior points minus two) always allows for compatible triangulations.

1 Introduction

Can any two planar point sets (that agree on the number of points and extreme points) be triangulated in the same manner? This intuitive question is the topic of the present paper. Apart from the theoretical interest in this basic problem, questions of this kind arise in various areas of application, including image analysis, morphing, and cartography.

To define the problem more carefully, let S_1 and S_2 be two finite sets of points in the Euclidean plane. Two triangulations T_1 of S_1 and T_2 of S_2 are called *compatible* if the face lattices formed by their triangles, edges, and vertices (points) are isomorphic. Compatible triangulations obviously exhibit isomorphic graphs of edges. The converse is not true, in general, as an edge isomorphism between T_1 and T_2 need not preserve the external face. This led us to prefer the notion of *compatible triangulations* to *isomorphic triangulations* which sometimes has been used in the literature.

⋆ Research supported by APART [Austrian Programme for Advanced Research and Technology] of the Austrian Acedemy of Sciences, and the EU Improving Human Potential Programme HPRI-1999-CT-00071
⋆⋆ Research supported by the FWF [Austrian Fonds zur Förderung der Wissenschaftlichen Forschung]
⋆⋆⋆ Research supported by Proyectos DGES-MEC PB98-0933 and Gen. Cat. 1999SGR00356

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 101–110, 2001.
© Springer-Verlag Berlin Heidelberg 2001

The problem of triangulating two given point sets compatibly comes in two flavors, namely where the bijection between the points of S_1 and S_2 is either fixed or variable. The case of fixed correspondence is a known problem and has been studied in Saalfeld [7]. He pointed out that compatible triangulations do not always exist and proposed several heuristic approaches for their construction. The challenging problem left open is to determine the complexity of the related decision problem, which so far is neither known to be NP-complete nor to be polynomially solvable.

The problem becomes easier if S_1 and S_2 represent the cyclic order of the vertices of two simple n-gons. Given a bijection between the polygon vertices (there are only n cyclic shifts to choose from) the existence can be decided in time $O(n^2)$, and compatible triangulability of the polygons can always be achieved by adding $O(n^2)$ extra points (so-called Steiner points) in either polygon; see Aronov et al. [2]. Improvements which take into account the number of reflex vertices [5] or the quality of the angles of the produced triangles [3] exist.

The present work is concerned with the problem of finding compatible triangulations *without* fixed point correspondence. To our knowledge this problem has not been studied before. The main question is, of course, under which propositions compatible triangulations do exist. It is clear that in both sets S_1 and S_2 the number n of points must be the same, as has to be the number h of points on their convex hulls $CH(S_1)$ and $CH(S_2)$, by the well-known formula $e = 3n - h - 3$ on the number of edges of a triangulation. Moreover, convex hull points necessarily map to convex hull points, and their cyclic order is preserved. This follows from the easily proven fact below: if two triangulations are compatible then any two corresponding triangles ijk and $i'j'k'$ must have the same orientation, either clockwise or counter-clockwise. (We ignore the symmetric case where the orientation is reversed for each matching pair of triangles.) Intuitively speaking, this means that the underlying point sets S_1 and S_2 'locally' exhibit the same *order type*, as defined in Goodman and Pollack [4]. It is interesting to note that the last two properties need *not* hold if only graph isomorphism rather than compatibility of the triangulations is required.

We will assume throughout that all point sets considered are in general position, meaning that no three points within a set lie on a common straight line. There are simple examples of point sets not in general position that do not admit compatible triangulations even if the propositions mentioned above are all met; see e.g. Krasser [6].

Throughout, let $|CH(S)|$ denote the number of extreme points in a point set S. Our main conjecture can be formulated in the following way.

Conjecture 1 *(Main Conjecture on Compatible Triangulations)*
Let S_1 and S_2 be two sets of points without fixed correspondence. Then compatible triangulations for S_1 and S_2 exist if these sets fulfill the following properties:

- $|S_1| = |S_2|$,
- $|CH(S_1)| = |CH(S_2)|$, *and*
- S_1 *and* S_2 *are in general position.*

We also consider an extended version of the compatible triangulation problem, where the correspondence between the points *on the convex hulls* (but not between the interior points) is prescribed. The status of the extended problem is still unresolved, too.

Conjecture 2 *(Extended Conjecture on Compatible Triangulations)*
Conjecture 1 still holds if the correspondence between the extreme points of S_1 and S_2 is given by a cyclically shifted labeling.

The motivation for stating the extended conjecture stems from the attempt to prove Conjecture 1 by induction, which naturally leads to assuming the stronger version as a hypothesis.

In this paper, first steps towards proving the conjectures above are undertaken. In fact, obtaining an affirmative answer would be a deep theoretical result, showing that all planar point sets of the same size and hull size are 'topologically equivalent' in this sense. The result, if true, gains in importance in view of the huge number of inequivalent order types for n points; see Aichholzer et al. [1] for the exact numbers of order types for $n \leq 10$.

Section 2 exhibits a family of point sets each of whose members allows for a compatible triangulation with any other point set (within the evident size and hull restrictions). In Section 3 we prove Conjecture 2 provided the number of interior points in either set is three or less. These restricted cases already reflect part of the intrinsic complexity of the problem: the orientation of the three interior points need to be reversed to allow for a solution in some cases. Section 4 shows that compatible triangulability can always be achieved by adding at most $k-2$ Steiner points, where k denotes the number of interior points. This contrasts the afore-mentioned situation for simple polygons [2], where a quadratic number of Steiner points is necessary in the worst case, and is a result of practical relevance. Finally, Section 5 concludes the paper with some related remarks.

2 A Family of Universal Sets

A point set U is called *weak universal* if for every point set S with $|S| = |U|$ and $|CH(S)| = |CH(U)|$ there is a triangulation of S compatible with a triangulation of U. The set U is called *universal* if, in addition, the correspondence between the extreme points of S and U may be prescribed by a cyclic shift. Conjecture 2, if true, states that all point sets are universal. Here we prove that certain point sets are indeed universal.
Let U be a point set with extreme points $u_1, \ldots, u_h$ (as they appear counterclockwise on the convex hull) and internal points $u_{h+1}, \ldots, u_n$ such that, for some $1 \leq m \leq h-2$,

- the subset $U \setminus \{u_1, \ldots, u_m\}$ is in convex position, and
- the internal set $\{u_{h+1}, \ldots, u_n\}$ lies to the left of both $u_h u_m$ and $u_1 u_{m+1}$; see Figure 1.

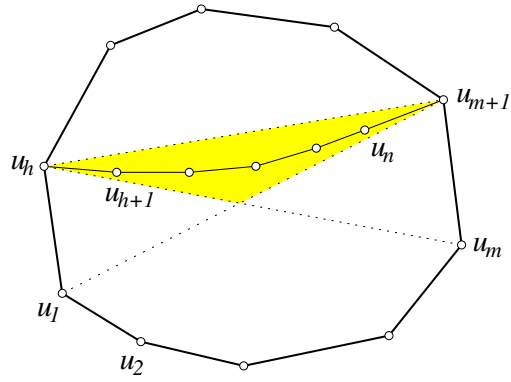


Fig. 1. The point set U.

We are going to prove that U is a universal set. The proof of the result requires a preliminary lemma.

Lemma 1 *Let P be a convex polygon with vertices $v_1, \ldots, v_h$ in counterclockwise order, and let I be an arbitrary point set interior to P. For any choice of indices $1 \leq i, j \leq h$ with $i \leq j - 2$ there exists a simple path π of edges such that*

(a) path π connects v_i, I, and v_j (in this order) and these are the only vertices of π;

(b) the simple polygon bounded by π and $v_i v_{i+1} \ldots v_j$ can be triangulated without using an edge that connects two vertices of π.

Proof. By induction on $d = j - i \geq 2$, the distance between v_i and v_j on P's boundary. For $d = 2$ the result is trivial: sort $I \cup \{v_i, v_j\}$ radially around v_{i+1}, to produce a path star-shaped as seen from v_{i+1}, and triangulate by connecting all points in I to this vertex.

Let now $d > 2$. If the set I is totally to the right of $v_{i+2}v_j$ then connect v_i and v_{i+2} and slice off P the triangle $v_i v_{i+1} v_{i+2}$ which cannot contain any point in I. Apply induction to the remaining convex polygon; the boundary distance between v_i and v_j is now $d - 1$.

If, on the other hand, there are points of I to the left of $v_{i+2}v_j$ then let p be the one maximizing the angle $v_i v_{i+1} p$; see Figure 2. Connect p to v_{i+1} and v_{i+2} and observe that the resulting triangle encloses no points of I. Now consider the convex polygon $Q = p v_{i+2} \ldots v_j$ (shaded in Figure 2). Let I' and I'' be the subsets of I exterior and interior to Q, respectively. By construction, no point of I' lies to the right of $v_{i+1}p$. So, as done for the case $d = 2$ above, construct the path π' for $I' \cup \{v_i, p\}$ that is star-shaped as seen from v_{i+1}, and build the corresponding triangulation. Finally, apply induction to the polygon Q whose boundary distance between p and v_j is $d - 1$. This yields a simple path π'' for

$I'' \cup \{p, v_j\}$ whose concatenation with π' obviously is still simple and satisfies the statement of the lemma. □

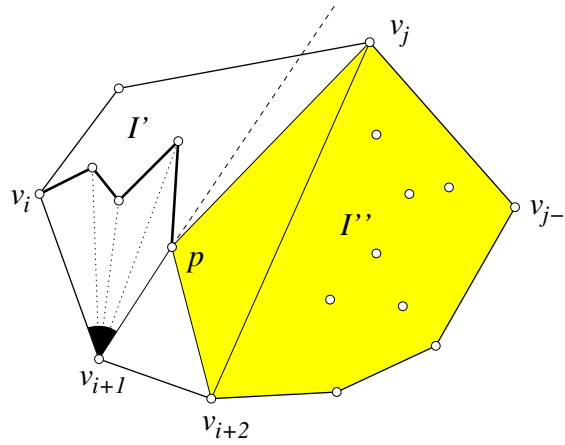


Fig. 2. Constructing the triangulation $T(\pi)$.

Theorem 1 *Let U be a set as described above. Then U is a universal set.*

Proof. Let S be any point set with $|S| = |U| = n$ and with extreme points $v_1, \ldots, v_h$ in counter-clockwise order. We construct for $CH(S)$ the path π from v_h to v_{m+1}, and the associated triangulation $T(\pi)$ on π's right-hand side, whose existence is given by Lemma 1. (Note that the index m is given from U and that the counter-clockwise boundary distance between v_h and v_{m+1} is at least two.) The crucial property is that $T(\pi)$ can be reproduced on the simple polygon $u_h \ldots u_n u_{m+1} u_m \ldots u_1$: only edges between non-consecutive vertices of the path $u_h \ldots u_n u_{m+1}$ lie outside this polygon, but those correspond to edges outruled by the definition of $T(\pi)$. It remains to be observed that any triangulation of $CH(S)$ on π's left-hand side can be put on top of the convex polygon $u_{m+1} u_{m+2} \ldots u_n$, making compatible the global triangulations. □

3 Few Interior Points

Below we prove Conjecture 2 for point sets with at most three interior points. Even in these seemingly simple situations, the illusion of a quick proof is destroyed by the fact that the orientation of the three interior points might have to be reversed to achieve compatibility. We first focus on the following class of point sets.

Lemma 2 *Let S be a point set containing an extreme point p such that $S \setminus \{p\}$ is in convex position. Then Conjecture 2 is true.*

PROOF. The set S fulfills the requirements for the set U in Section 2 for $m = 1$ and thus is a universal set by Theorem 1. □

Theorem 2 *Let S_1 and S_2 be two n-point sets each of which contains $k \leq 3$ interior points. Then Conjecture 2 is true.*

PROOF. The cases $k = 0$, where S_1 and S_2 are in convex position, and $k = 1$, where star-like compatible triangulations always exist, are trivial.

The case $k = 2$ is still easy. It is always possible to find two extreme points of S_1 labeled a and b such that (1) a and b lie on different sides of the line through the two interior points of S_1, and (2) the (extreme) points of S_2 labeled a and b lie on different sides of the line through the two interior points of S_2. (The non-existence of such a pair a, b would imply that one of S_1 and S_2 totally lies on one side of the respective line.) We build two triangles for S_1 by joining its interior points x and y and connecting a and b to both of them. The remaining extreme points can be connected to either x or y accordingly. Doing the same for S_2 results in compatible triangulations.

The case $k = 3$ is more intriguing. Let us refer to the triangle formed by the three interior points of S_1 (or S_2) as the *interior triangle* of S_1 (or S_2). We distinguish between two situations, the former being similar to the case $k = 2$.

(a) S_1 as well as S_2 contains extreme points with labels a, b, c each of which allows for a triangle adjacent to the respective interior triangle such that no overlap of these four triangles occurs. In this case, we integrate these triangles for both sets. For extreme points with labels intermediate to a, b, and c, edges can be drawn now to a unique interior point, in a compatible manner and much like the case $k = 2$.

(b) Otherwise. The solution we are going to construct will not contain the interior triangle. (In fact, no such solution exists.) Let xyz be this triangle for S_1. Let D_x be the subset of S_1 not on x's side of the line through y and z. Define subsets D_y and D_z analogously; see Figure 3(a). Since labels as above cannot be found, the subsets of S_2 that correspond to D_x, D_y, and D_z have to look as in Figure 3(b), where two of them are totally contained in the same trilateral cell. Without loss of generality, let the third subset correspond to D_z. We wish to redefine D_x, D_y, and D_z so as to be disjoint sets, by introducing 'breakpoint' labels a, b, and c such that D_x ranges from $a + 1$ to b, D_y ranges from $b + 1$ to c, and D_z ranges from $c + 1$ to a. This is always possible unless two of the sets are singletons consisting of the same point p. In this case, however, removal of p leaves the set $S_1 \setminus \{p\}$ in convex position such that compatible triangulations exist by Lemma 2. Otherwise, we assign labels x, y, and z to the interior triangle of S_2 and draw compatible edges as shown in Figure 4. Note that the orientation of xyz has been reversed for S_2. The two shaded polygons for S_1 are convex and thus can be triangulated compatibly with their counterparts for S_2. It remains to partition the still untriangulated area for S_1 into convex parts. Possible reflex

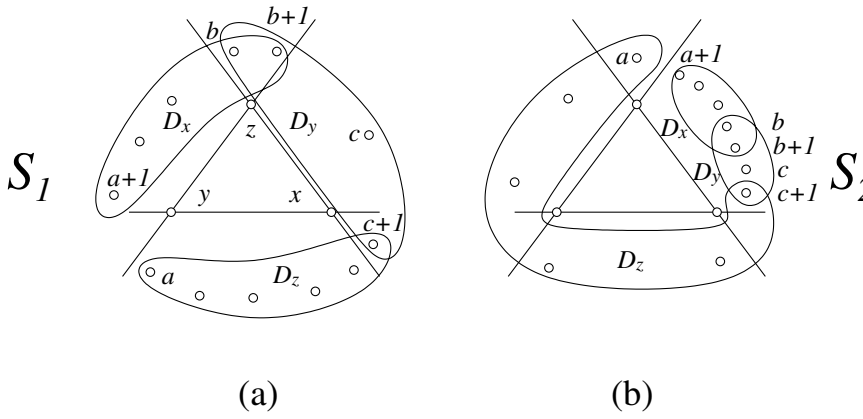


Fig. 3. Two subsets move into the same cell.

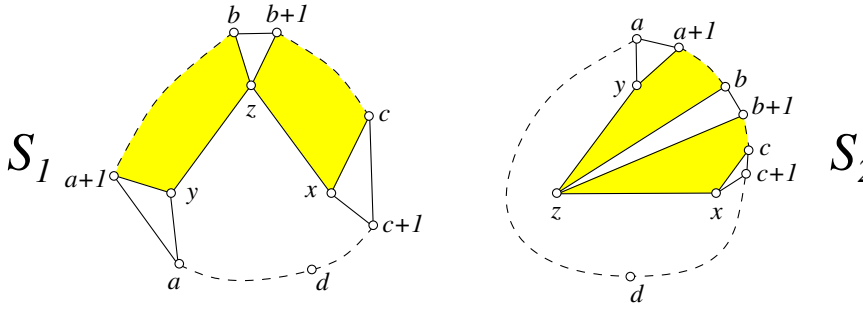


Fig. 4. Compatible edge skeletons that yield convex parts.

angles may occur at x or y. If, as in Figure 4, this happens for x then there must exist a point labeled d in $D_z \setminus D_y$ which makes this angle convex. (If d would not exist then all extreme points of S_2 would lie on the same side of the line through x and z.) The situation for y is similar. This completes the construction of compatible triangulations. □

4 When Steiner Points Are Allowed

Since it is still left open whether Conjectures 1 and 2 are true, the question of whether compatibility can be always forced by adding a reasonably small number of Steiner points suggests itself. We demonstrate below that a linear number suffices. An efficient algorithm for finding compatible triangulations is implicit in the proof.

Let v be an extreme point of the set S. The *ear* $E(v)$ of v is the triangle spanned by the two edges of $CH(S)$ incident to v. An ear is called *empty* if it does not enclose any (interior) point of S.

Theorem 3 *Let S_1 and S_2 be two n-point sets whose $n-k$ extreme points are in fixed correspondence. If $k \geq 2$ then S_1 and S_2 can be triangulated compatibly by introducing at most $k-2$ Steiner points.*

Proof. By induction on k, the number of interior points. Let $\sigma(k)$ be the number of Steiner points needed. We trivially have $\sigma(2) = \sigma(3) = 0$ by Theorem 2, so let us assume $k \geq 4$.

Let $v_1, \ldots, v_h$ be the extreme points of S_1, and $w_1, \ldots, w_h$ the corresponding extreme points of S_2, in counter-clockwise order. We may assume that for no index i the ears $E(v_i)$ and $E(w_i)$ are simultaneously empty. Otherwise, compatible edges $v_{i-1}v_{i+1}$ and $w_{i-1}w_{i+1}$ are introduced, and the sets $S_1 \setminus \{v_i\}$ and $S_2 \setminus \{w_i\}$ are considered. It remains to discuss the cases below.

Case (a) There is an index i such that both ears $E(v_i)$ and $E(w_i)$ are non-empty. Let $p \in S_1$ be the first point hit when rotating the edge $v_{i-1}v_i$ counter-clockwise around v_{i-1}. As $E(v_i)$ contains points, p must be an interior point. So by introducing one Steiner point s slightly to the left of $v_{i-1}p$ and close to edge v_iv_{i+1}, it is possible to create an empty triangle v_ips and three adjacent empty triangles as shown in Figure 5(a). Obviously, the same construction can be repeated for the ear $E(w_i)$ using a suitable Steiner point t, giving four compatible triangles. We now apply induction to the sets $S_1 \cup \{s\} \setminus \{v_i\}$ and $S_2 \cup \{t\} \setminus \{w_i\}$ whose number of interior points is $k-1 \geq 3$. This implies $\sigma(k) \leq \sigma(k-1)+1 \leq (k-1)-2+1 = k-2$.

Case (b) For each index i, exactly one of the ears $E(v_i)$ and $E(w_i)$ is empty. Clearly S_1 (and S_2) has $h \geq 4$ extreme points in this case. Let us assume $h \geq 5$ first. Then one of the sets, S_1 say, has at least three empty ears. Choose two of them which do not overlap, $E(v_i)$ and $E(v_j)$, for $j \geq i+2$. Consider the convex polygon $CH(S_1) \setminus (E(v_i) \cup E(v_j))$, and some triangulation T_1 thereof which uses only edges which separate $E(v_i)$ and $E(v_j)$. Let T_2 denote the compatible triangulation of the corresponding polygon for S_2. Now sweep T_1, triangle by triangle, with a line ℓ_1 starting at position $v_{i+1}v_{i-1}$ and stopping at position $v_{j-1}v_{j+1}$. When sweeping a triangle $v_av_bv_{b+1}$ we require ℓ_1 to pass through point v_a and to cross the edge v_bv_{b+1}. Sweep T_2 with a line ℓ_2 in a similar manner. Let α_1 be the current number of interior points of S_1 to the left of ℓ_1 (same for α_2). Since $E(v_i)$ and $E(v_j)$ are empty ears we have $\alpha_1 = 0$ in the beginning and $\alpha_1 = k$ at the end. On the other hand, $\alpha_2 \geq 1$ in the beginning and $\alpha_2 \leq k-1$ at the end, because $E(w_i)$ and $E(w_j)$ are both non-empty. As a consequence, there exist indices a and b that yield $\alpha_1 = \alpha_2$ while ℓ_1 sweeps the triangle $v_av_bv_{b+1}$ and ℓ_2 sweeps the corresponding triangle $w_aw_bw_{b+1}$. Now pick a Steiner point s on (this very position of) ℓ_1 and close to v_bv_{b+1}, and put in edges from s to v_b, v_{b+1}, and v_a; see Figure 5(b). Repeat the construction for S_2 which gives a compatible triangle. Finally, apply induction to the respective subsets of S_1 and S_2 on either side of the lines. They have k' and k'' interior

points, respectively, for $1 \leq k', k'' \leq k-1$. We may assume $k' \geq 2$ because $k \geq 4$. We get $\sigma(k) \leq \sigma(k') + \sigma(k'') + 1$, with $\sigma(k') \leq k' - 2$ and $\sigma(k'') \leq k'' - 1$ and equality in the last term if $k'' = 1$. In summary, $\sigma(k) \leq k - 2$ again.

The case $h = 4$ which has not been covered yet is easier. There is a line ℓ_1 crossing the edges v_1v_2 and v_3v_4 which has exactly 2 interior points of S_1 to its left and $k - 2 \geq 2$ such points to its right. There is also a similar line ℓ_2 that crosses the corresponding edges for S_2. Pick two Steiner points on ℓ_1, one close to v_1v_2 and the other close to v_3v_4, and create the two respective triangles, as shown in Figure 5(c). Repeat the construction for ℓ_2 and S_2 which results in two compatible triangles. Now apply induction to the respective subsets of S_1 and S_2 on either side of the lines. This gives $\sigma(k) \leq \sigma(2)+\sigma(k-2)+2 \leq (k-4)+2 = k-2$ which completes the argumentation. □

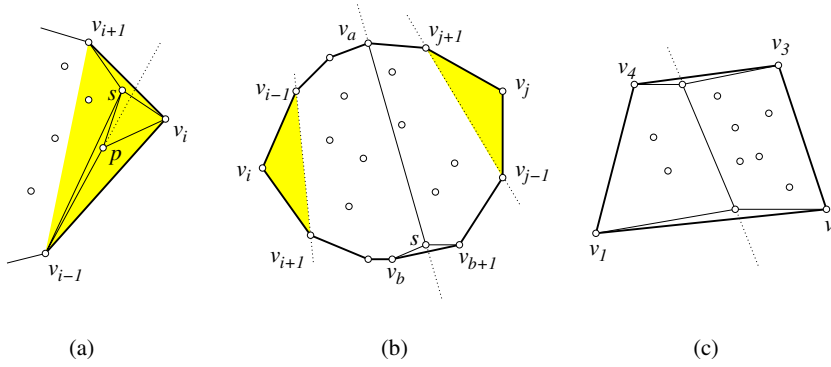


Fig. 5. Placing Steiner points.

5 Remarks

It is a challenging open problem to prove our conjectures for general point sets. Unfortunately, the constructions in the proof of Theorem 2 do not seem to generalize easily to the case of more interior points. Approaches to prove Conjecture 2 by induction suffer from the lack of appropriate methods for splitting the problem. Here is a discouraging related fact: given two quadrilaterals, one convex and the other star-shaped, plus one interior point for each, it may not be possible to triangulate them compatibly if the correspondence between their boundary vertices is fixed by a cyclic shift.

We tested Conjecture 2 for small point set sizes, utilizing a database of combinatorially inequivalent point sets (i.e., order types); see [1]. If two point sets exhibit the same order type then every triangulation of one set also leads to a compatible triangulation for the other. By checking all different pairs of order types we could verify the conjecture for up to 8 points. Although our database provides all order types of size up to 10, exhaustive tests turned out to be time

consuming and are still incomplete for size 9. In fact, determining the time complexity of computing compatible triangulations, if they exist, for two given n-point sets is an open problem.

Finally, we pose the problem of triangulating compatibly two point sets in *fixed* correspondence when Steiner points are allowed. In view of the applications mentioned in [7], a fast algorithm using a small number of Steiner points would be of practical relevance.

References

1. O.Aichholzer, F.Aurenhammer, H.Krasser, *Enumerating order types for small point sets with applications.* 17^{th} Ann. ACM Symp. Computational Geometry, Medford, MA, 2001 (to be presented).
2. B.Aronov, R.Seidel, D.Souvaine, *On compatible triangulations of simple polygons.* Computational Geometry: Theory and Applications 3 (1993), 27-35.
3. M.Bern, A.Sahai, *Isomorphic triangulations and map conflation.* Personal communication, 2000.
4. J.E.Goodman, R.Pollack, *Multidimensional sorting.* SIAM J. Computing 12 (1983), 484-507.
5. E.Kranakis, J.Urrutia, *Isomorphic triangulations with small number of Steiner points.* Int'l J. Computaional Geometry & Applications 9 (1999), 171-180.
6. H.Krasser, *Kompatible Triangulierungen ebener Punktmengen.* Master Thesis, Institute for Theoretical Computer Science, Graz University of Technology, Graz, Austria, 1999.
7. A.Saalfeld, *Joint triangulations and triangulation maps.* Proc. 3^{rd} Ann. ACM Sympos. Computational Geometry, Waterloo, Canada, 1987, 195-204.

An Improved Upper Bound on the Size of Planar Convex-Hulls*

Abdullah N. Arslan and Ömer Eğecioğlu

Department of Computer Science
University of California, Santa Barbara
CA 93106, USA
{arslan,omer}@cs.ucsb.edu

Abstract. Let $\mathcal{C}$ be the convex-hull of a set of points S with integral coordinates in the plane. It is well–known that $|\mathcal{C}| \leq cD^{2/3}$ for some constant c where D is the diameter of S: i.e. the maximum distance between any pair of points in S. It has been shown that $c = 7.559..$ for an arbitrary S, and $c = 3.496..$ in the special case when S is a ball centered at the origin in the plane. In this paper we show that $c = 12/\sqrt[3]{4\pi^2} = 3.524..$ is sufficient for an arbitrary set of lattice points S of diameter D in the plane, and $|\mathcal{C}| \sim 12\sqrt[3]{2/(9\pi^2)}\, D^{2/3} = 3.388..D^{2/3}$ is achieved asymptotically. Our proof is based on the construction of a special set in first quadrant, and the analysis of the result involves the calculation of the average order of certain number-theoretical functions associated with the Euler totient function $\phi(n)$.

1 Introduction

A lattice point is a point with integral coordinates. Given a set S of lattice points in the plane, let $\mathcal{C}$ be the convex-hull of S, and denote the number of extreme points in $\mathcal{C}$ by $|\mathcal{C}|$. The behavior of $|\mathcal{C}|$ as a function of parameters associated with S has been studied in various contexts in computational geometry, computer graphics [10,6,9], and integer programming [1,3,5].

Andrews' general theorem [1] on convex bodies in d-dimensional space implies that in the plane $|\mathcal{C}|$ is bounded by

$$|\mathcal{C}| \leq cD^{2/3} \tag{1}$$

for some constant c, where D is the diameter of S: i.e. the maximum distance between any two points in S. Another proof of this bound for the plane was given by Katz and Volper [9]. The constant c has also been studied. Balog and Bárány [2] showed that when S is the ball of radius $r = D/2$ centered at the origin, i.e. $S = rB^2$, then

$$0.207..D^{2/3} \leq |\mathcal{C}| \leq 3.496..D^{2/3}.$$

* Supported in part by NSF Grant No. CCR–9821038, and a UCSB-COR faculty research grant.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 111–120, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Therefore one can take $c = 3.496..$ in the special case when S is a ball of diameter D. A d-dimensional analysis for the ball appears in [4]. Har-Peled [7] showed that the value $c = 6\sqrt[3]{2} = 7.559..$ is sufficient for the bound (1) in the plane for arbitrary S.

In this paper we investigate further the nature of the constant c in (1) and show that

$$c = 12/\sqrt[3]{4\pi^2} = 3.524..$$

suffices for an arbitrary S of diameter D in the plane. Our proof is based on the construction of a special set in first quadrant of the plane which satisfies certain constraints. The construction involves selecting a set of fractions in a particular order as slopes of the line segments of the convex-hull. Based on the properties of the average order of certain number-theoretical functions associated with the Euler totient function $\phi(n)$, we derive an upper bound on the size of the set constructed. This leads to an improved value for c. We also show that using the construction idea of the proof, we can always create a convex-hull $\mathcal{C}$ with a given diameter D such that

$$|\mathcal{C}| \sim 12\sqrt[3]{2/(9\pi^2)}\, D^{2/3} = 3.388..D^{2/3} \tag{2}$$

is achieved asymptotically.

The organization of this paper is as follows. Section 2 gives the number-theoretical background we require, and includes the proof of the main theorem, which we then use in Section 3 for proving our result on the improved value of c. In Section 4, we construct large convex-hulls with a given diameter proving (2).

2 Number-Theoretical Definitions and Results

We use the classical book by Hardy and Wright [8] as our main reference for the definitions and basic results used in this section. We denote by (r, s) the greatest common divisor of r and s.

- The *Euler totient function* $\phi(n)$ ([8], p. 52) is defined as follows:
  1. $\phi(1) = 1$
  2. for $n > 1$, $\phi(n)$ is the number of positive integers less than n and relatively prime to n.
- The *Möbius function* $\mu(n)$ ([8], p. 234) is defined by
  1. $\mu(1) = 1$,
  2. $\mu(n) = 0$ if n has a square factor,
  3. $\mu(p_1 p_2 \cdots p_k) = (-1)^k$ if all the primes $p_1, p_2, \ldots, p_k$ are different.
- The *Riemann zeta function* $\zeta(s)$ ([8], p. 245) is defined for $s > 1$ by

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \ .$$

We use the following well-known results:

Lemma 1.

$$\phi(n) = \sum_{d|n} \frac{n}{d}\mu(d), \tag{3}$$

$$\zeta(2)^{-1} = \sum_{d=1}^{\infty} \frac{\mu(d)}{d^2} = \frac{6}{\pi^2}\ . \tag{4}$$

Proof. The proof of (3) is in ([8], p. 235). The proof of (4) can be found in ([8], Thm. 287, p. 250; Thm. 293, p. 251).

Let $\Phi_0(n) = \phi(1) + \phi(2) + \cdots + \phi(n)$. It is known in relation to Farey fractions ([8], Thm. 330, p. 268) that

$$\Phi_0(n) = \frac{3n^2}{\pi^2} + O(n \log n), \tag{5}$$

so that the *average order* (([8], p. 263) of the function $\phi(n)$ is given by (5). Theorem 1 below gives an expression for the average order of the function $n^r\phi(n)$ for $r \geq 0$, generalizing (5).

Theorem 1. *For any integer* $r \geq 0$

$$\Phi_r(n) = 1^r \cdot \phi(1) + 2^r \cdot \phi(2) + \cdots + n^r \cdot \phi(n) = \frac{6}{(r+2)\pi^2} n^{r+2} + O(n^{r+1} \log n)\ . \tag{6}$$

Proof. By (3) of lemma 1 we have

$$\Phi_r(n) = \sum_{m=1}^{n} m^r \phi(m) = \sum_{m=1}^{n} m^r \sum_{d|m} \frac{m}{d}\mu(d) = \sum_{m=1}^{n} \sum_{dd'=m} m^r d'\mu(d)$$

$$= \sum_{m=1}^{n} \sum_{dd'=m} d^r (d')^{r+1}\mu(d) = \sum_{dd' \leq n} d^r (d')^{r+1}\mu(d) = \sum_{d=1}^{n} d^r \mu(d) \sum_{d'=1}^{\lfloor n/d \rfloor} (d')^{r+1}$$

Using the fact that

$$\sum_{k=1}^{n} k^{r+1} = \frac{n^{r+2}}{r+2} + O(n^{r+1}),$$

we get

$$\sum_{d'=1}^{\lfloor n/d \rfloor} (d')^{r+1} = \frac{1}{r+2}\left(\frac{n}{d}\right)^{r+2} + O\left(\frac{n^{r+1}}{d^{r+1}}\right).$$

Therefore

$$\Phi_r(n) = \frac{n^{r+2}}{r+2} \sum_{d=1}^{n} \frac{\mu(d)}{d^2} + O\left(n^{r+1} \sum_{d=1}^{n} \frac{1}{d}\right)$$

$$= \frac{n^{r+2}}{r+2} \sum_{d=1}^{\infty} \frac{\mu(d)}{d^2} + O\left(n^{r+2} \sum_{d=n+1}^{\infty} \frac{1}{d^2}\right) + O\left(n^{r+1} \log n\right)$$

$$= \frac{n^{r+2}}{r+2}\zeta(2)^{-1} + O(n^{r+1}) + O(n^{r+1} \log n)\ ,$$

and the theorem follows from the last equality and (4) of lemma 1.

We need the following result relating $\Phi_0(n)$ and $\Phi_1(n)$ as a step in our study of $|\mathcal{C}|$.

Theorem 2. *Let* $\Phi_0(n) = \phi(1) + \phi(2) + \cdots + \phi(n)$,
$\Phi_1(n) = 1 \cdot \phi(1) + 2 \cdot \phi(2) + \cdots + n \cdot \phi(n)$.

Then

$$\Phi_0(n) \sim \frac{3}{\sqrt[3]{4\pi^2}} \Phi_1(n)^{2/3} = 0.8810516..\Phi_1(n)^{2/3}. \tag{7}$$

Proof. The theorem follows by combining the expressions for $\Phi_0(n)$ and $\Phi_1(n)$ obtained as the cases $r = 0$, and $r = 1$ of theorem 1.

We note that also the magnitude of the error term in (7) can be calculated by using the full expressions for $\Phi_0(n)$ and $\Phi_1(n)$ obtained. This gives

$$\Phi_0(n) = \frac{3}{\sqrt[3]{4\pi^2}} \Phi_1(n)^{2/3} + O\left(\Phi_1(n)^{1/3} \log \Phi_1(n)\right).$$

We omit the details of this calculation.

3 An Improved Upper Bound

We first establish an upper bound on the size of the convex-hull of lattice points in first quadrant of the plane in the following lemma.

Lemma 2. *Let* $S_{a,b}$ *be a set of lattice points in first quadrant of the xy-plane enclosed by* $y = a$ *and* $x = b$*. For the convex-hull* $\mathcal{C}_{a,b}$ *of* $S_{a,b}$*, there holds*

$$|\mathcal{C}_{a,b}| \leq \frac{3}{\sqrt[3]{4\pi^2}} (a+b)^{2/3} + O\left((a+b)^{1/3} \log (a+b)\right)$$

Proof. Without loss of generality, $S_{a,b}$ includes the points $(0, a)$ and $(b, 0)$ and both a and b are positive integers (Figure 1). Instead of the point-set $\mathcal{C}_{a,b}$ we consider the corresponding set of slopes $R_{a,b}$ of the line segments (edges) connecting the consecutive extreme points of the convex-hull. Clearly, $|\mathcal{C}_{a,b}| = |\mathcal{R}_{a,b}| + 1$, so we can alternately study the properties of the set $\mathcal{R}_{a,b}$. By convexity, the slopes of the edges of the convex-hull are all different. Furthermore $S_{a,b}$ is a set of lattice points, and therefore the slopes of the non-vertical edges are all rational numbers.

Let $\mathcal{R}^*_{a+b}$ be an optimal set for the following problem: maximize $|\mathcal{R}|$ subject to

$$\sum_{\frac{y}{x} \in \mathcal{R}} y \; + \; \sum_{\frac{y}{x} \in \mathcal{R}} x \; = \; \sum_{\frac{y}{x} \in \mathcal{R}} y + x \; \leq \; a + b \tag{8}$$

Clearly, for any a and b, $|\mathcal{R}_{a,b}| \leq |\mathcal{R}^*_{a+b}|$. We will find a bound for $|\mathcal{R}^*_{a+b}|$. Let $\mathcal{Q}_i$ for $i \geq 0$ be the set of slopes defined as follows:

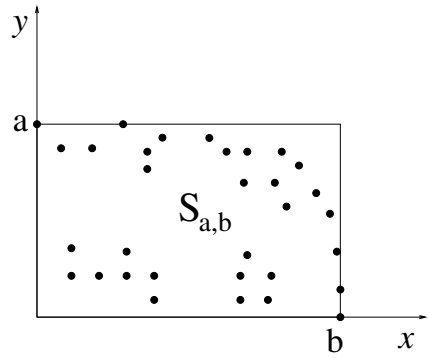


Fig. 1. A set of lattice points $S_{a,b}$ in first quadrant.

1. $\mathcal{Q}_0 = \emptyset$,
2. $\mathcal{Q}_1 = \left\{\frac{0}{1}, \frac{1}{0}\right\}$. We assume that the fraction $\frac{1}{0}$ is defined as a slope and it represents the vertical edge whose length is one unit.
3. For $i > 1$, $\mathcal{Q}_i = \left\{\frac{y}{x} \mid y + x = i \text{ and } (y, x) = 1\right\}$.

Table 1 illustrates the first few values $\mathcal{Q}_i$, $\phi(i)$, $\Phi_0(i)$, and $\Phi_1(i)$. The following properties can easily be seen for $i > 1$:

$$|\mathcal{Q}_i| = \phi(i) \tag{9}$$

$$\sum_{\frac{y}{x} \in \mathcal{Q}_i} y + x = i \; |\mathcal{Q}_i| \tag{10}$$

Let $n+1$ be the smallest number such that $\sum_{i=1}^{n+1} i|\mathcal{Q}_i| > a + b$, and $\mathcal{Q}'_{n+1}$ be an arbitrary subset of $\mathcal{Q}_{n+1}$ such that

$$|\mathcal{Q}'_{n+1}| = \left\lfloor \frac{(a+b) - \sum_{i=1}^{n} i|\mathcal{Q}_i|}{n+1} \right\rfloor$$

Then consider the following set

$$\mathcal{R}_{a+b} = \left(\bigcup_{i=1}^{n} \mathcal{Q}_i \right) \cup \mathcal{Q}'_{n+1}. \tag{11}$$

We claim that $\mathcal{R}_{a+b}$ is a maximal set which satisfies the constraint (8). The expression (11) describes a greedy construction : To include in set $\mathcal{R}_{a+b}$, select a fraction whose numerator-denominator sum is the smallest. Continue including fractions until the sum of all the numerators and the denominators of the fractions currently in the set exceeds $a + b$. Since $\mathcal{R}_{a+b}$ is a set, no fraction can be included in the set more than once, and since the slopes are all different, among the equivalent fractions, the irreducible one has the smallest numerator-denominator sum. This explains why only the relatively prime numbers are to be considered in the construction.

Table 1. Table of $\mathcal{Q}_i$, $\phi(i)$, $\Phi_0(i)$, and $\Phi_1(i)$ for $i = 1, 2, \ldots, 7$.

i	$\mathcal{Q}_i$	$\phi(i)$	$\Phi_0(i)$	$\Phi_1(i)$
1	$\{\frac{0}{1}, \frac{1}{0}\}$	1	1	1
2	$\{\frac{1}{1}\}$	1	2	3
3	$\{\frac{1}{2}, \frac{2}{1}\}$	2	4	9
4	$\{\frac{1}{3}, \frac{3}{1}\}$	2	6	17
5	$\{\frac{1}{4}, \frac{2}{3}, \frac{3}{2}, \frac{4}{1}\}$	4	10	37
6	$\{\frac{1}{5}, \frac{5}{1}\}$	2	12	49
7	$\{\frac{1}{6}, \frac{2}{5}, \frac{3}{4}, \frac{4}{3}, \frac{5}{2}, \frac{6}{1}\}$	6	18	91

From (11) we have

$$\bigcup_{i=1}^{n} \mathcal{Q}_i \;\subseteq\; \mathcal{R}_{a+b} \;\subseteq\; \bigcup_{i=1}^{n+1} \mathcal{Q}_i \tag{12}$$

Using (12) and (9) we find that

$$\sum_{i=1}^{n} \phi(i) \;+\; 1 \;\leq\; |\mathcal{R}_{a+b}| \;\leq\; \sum_{i=1}^{n+1} \phi(i) \;+\; 1 \tag{13}$$

Again using (12) and (9) together with the expression (10), and using the fact that by construction $n+1$ is the smallest number such that $\sum_{i=1}^{n+1} i|\mathcal{Q}_i| > a+b$ we get

$$\sum_{i=1}^{n} i\phi(i) \;+\; 1 \;\leq\; a+b \;\leq\; \sum_{i=1}^{n+1} i\phi(i) \;+\; 1 \tag{14}$$

We note that $\phi(n+1) = O(\Phi_1(n)^{1/3})$ using the expression $\Phi_1(n)$ obtained from (6) with $r = 1$. Therefore we can write the following upper bound for $|\mathcal{R}_{a+b}|$ using (13):

$$|\mathcal{R}_{a+b}| = \sum_{i=1}^{n} \phi(i) \;+\; O\left(\Phi_1(n)^{1/3}\right) = \Phi_0(n) + O\left(\Phi_1(n)^{1/3}\right)$$

Furthermore from Theorem 2

$$|\mathcal{R}_{a+b}| = \frac{3}{\sqrt[3]{4\pi^2}}\Phi_1(n)^{2/3} + O\left(\Phi_1(n)^{1/3}\log\Phi_1(n)\right) + O\left(\Phi_1(n)^{1/3}\right), \tag{15}$$

and from (14) we obtain that $\Phi_1(n) \sim a+b$. Therefore

$$|\mathcal{R}_{a+b}| = \frac{3}{\sqrt[3]{4\pi^2}}\;(a+b)^{2/3} + O\left((a+b)^{1/3}\log(a+b)\right)$$

which proves the bound for $|\mathcal{C}_{a,b}|$ of the lemma.

Theorem 3. *For the convex-hull $\mathcal{C}$ of a set S of lattice points of diameter D in the plane, there holds*

$$|\mathcal{C}| \leq \frac{12}{\sqrt[3]{4\pi^2}}\, D^{2/3} + O\left(D^{1/3}\log D\right)$$

Proof. S can be enclosed by an axis-parallel rectangle whose sides are at most D in length. Using this rectangle, we can partition S into four parts such that each part is confined to a single quadrant as shown in Figure 2. For the size of $\mathcal{C}$ we have

$$|\mathcal{C}| \leq |\mathcal{C}_{a_1,b_1}| + |\mathcal{C}_{a_2,b_2}| + |\mathcal{C}_{a_3,b_3}| + |\mathcal{C}_{a_4,b_4}|$$

By Lemma 2 we can rewrite the inequality as

$$|\mathcal{C}| \leq \frac{3}{\sqrt[3]{4\pi^2}}((a_1+b_1)^{2/3}+(a_2+b_2)^{2/3}+(a_3+b_3)^{2/3}+(a_4+b_4)^{2/3})+O(D^{1/3}\log D) \tag{16}$$

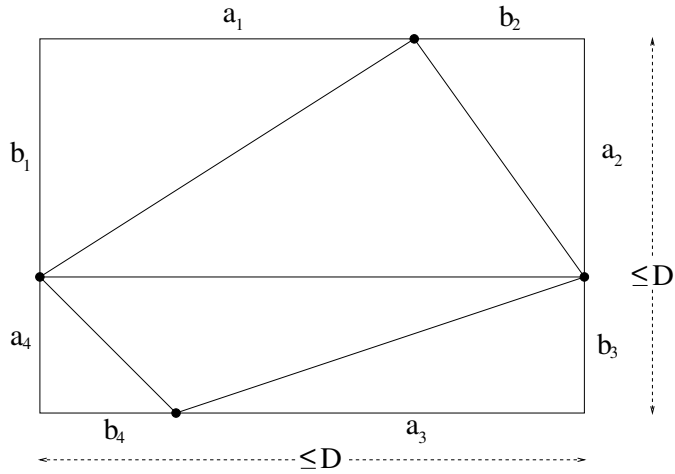


Fig. 2. Partitions of S in quadrants.

The function $f(x) = x^{2/3}$ is concave on $[0, \infty)$. In particular for nonnegative x_1 and x_2,

$$x_1{}^{2/3} + x_2{}^{2/3} \leq 2\left(\frac{x_1+x_2}{2}\right)^{2/3}.$$

Therefore, the equality in (16) becomes

$$|\mathcal{C}| \leq \frac{3}{\sqrt[3]{4\pi^2}}\; 4\left(\frac{a_1+b_1+a_2+b_2+a_3+b_3+a_4+b_4}{4}\right)^{2/3} \;+\; O\left(D^{1/3}\log D\right)$$

Since a_1+b_2, a_3+b_4, a_2+b_3, and a_4+b_1 are all smaller than or equal to D, the inequality for $|\mathcal{C}|$ of the theorem holds.

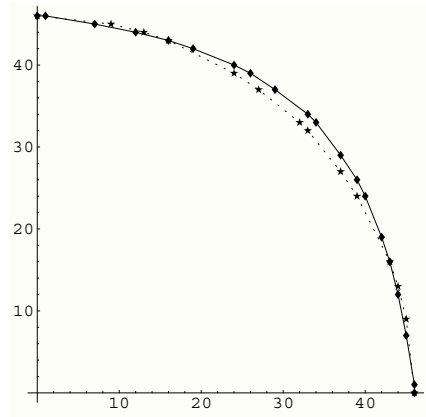


Fig. 3. $n = 7$, $a = b = 46 = \frac{1}{2}\Phi_1(7) + \frac{1}{2}$. The dotted line is used for the convex-hull of the set of lattice points in the ball of radius 46 in first quadrant. The solid lines show the constructed hull $\mathcal{C}(7)$.

4 Constructing a Large Convex-Hull with a Given Diameter

To investigate how tight the new value of c in the upper bound for $|\mathcal{C}|$ is, we revisit the proof of Lemma 2. We first describe how we can actually construct a convex-hull $\mathcal{C}_{a+b}$ from the set of slopes $\mathcal{R}_{a+b}$ created in the proof for particular a and b. More precisely, for every n, we construct a convex-hull $\mathcal{C}(n)$ in first quadrant with $a = b = \frac{1}{2}\Phi_1(n) + \frac{1}{2}$, symmetric about the line $y = x$. As its set of slopes, we take

$$\mathcal{R}(n) = \bigcup_{i=1}^{n} \mathcal{Q}_i.$$

$\mathcal{C}(n)$ connects the point $(0, \frac{1}{2}\Phi_1(n) + \frac{1}{2})$ on the y-axis to $(\frac{1}{2}\Phi_1(n) + \frac{1}{2}, 0)$ on the x-axis and consists of $\Phi_0(n)$ edges. We start with the point $(0, \frac{1}{2}\Phi_1(n) + \frac{1}{2})$ as the first vertex, and if the i-th smallest fraction in $\mathcal{R}(n)$ is y_i/x_i, then we place the next vertex on the hull x_i units to the right and y_i units down from the current vertex. See Figure 4 for the $n = 5$ case.

Now consider the $\mathcal{C}(n)$ constructed as we described. If it includes a vertex whose distance from the origin is larger than $\frac{1}{2}\Phi_1(n) + \frac{1}{2}$, then the diameter of the set constructed by taking four copies of $\mathcal{C}(n)$ around the origin will have diameter larger than D. That this is indeed the case can be seen from Figure 3, where the point farthest from the origin in our construction is not inside the ball with radius 46. Thus the constant $c = 12/\sqrt[3]{4\pi^2} = 3.352..$ in the upper bound is not tight.

We next investigate how close we can come to this value. It follows by symmetry and convexity that the farthest vertex from the origin on $\mathcal{C}(n)$ constructed is one of the endpoints of the edge with slope 1 on the hull as shown in Figure

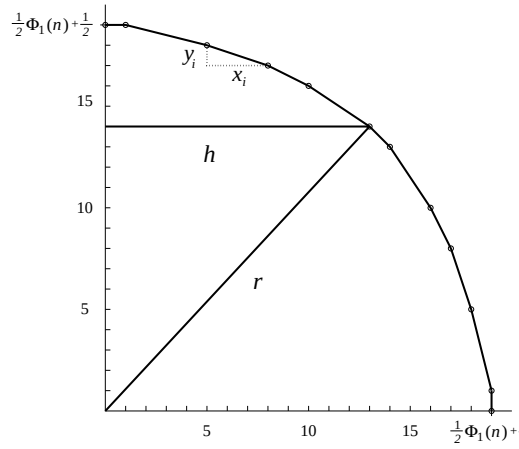


Fig. 4. $n = 5$, $a(n) = b(n) = \frac{1}{2}\Phi_1(5) + \frac{1}{2} = 19$.

4. Denote by $h(n)$ the x-coordinate of this point, and by $r(n)$ its distance from the origin. By construction

$$h(n) \;=\; \sum_{i+j\le n} j\chi[(i,j)=1]\chi[i\le j] \;=\; \sum_{j=1}^{n} j \sum_{i=1}^{j} \chi[(i,j)=1]\chi[i+j\le n] \quad (17)$$

where χ is the indicator function of its argument, i.e., it is 1 if its argument evaluates to true, and 0 otherwise.

We next show that $h(n) \sim \frac{3}{8}\Phi_1(n)$. Since intercepts of the hull are $a(n) = b(n) = \frac{1}{2}\Phi_1(n) + \frac{1}{2}$ this implies $h(n) \sim \frac{3}{4}a(n)$, and we can calculate $r(n)$ by the Pythagorean formula as

$$r(n) \sim \frac{3\sqrt{2}}{4}a(n). \quad (18)$$

Theorem 4. *Suppose $h(n)$ is as defined in (17), and $\Phi_1(n)$ as defined in Theorem 2. Then $h(n) = \frac{3}{8}\Phi_1(n) + O(n^2 \log n)$.*

Proof. We will indicate the derivation of the asymptotic part and ignore the calculation of the error terms. First we write $h(n) = h_1(n) + h_2(n)$ where in $h_1(n)$, the index j runs from 1 to $n/2$, and in h_2, j runs from $n/2+1$ to n. Then

$$h_1(n) = \sum_{j=1}^{\frac{n}{2}} j \sum_{i=1}^{j} \chi[(i,j)=1],$$

since for the indices in question $i \le j$ and we automatically have $i + j \le n$. Therefore $h_1(n) = \Phi_1(\frac{n}{2})$. On the other hand, if $j \ge n/2$, then $n - j \le j$ and

$$h_2(n) = \sum_{j=\frac{n}{2}}^{n} j \sum_{i=1}^{j} \chi[(i,j)=1]\chi[i\le n-j]$$

$$\sim \sum_{j=\frac{n}{2}}^{n} j \cdot \frac{n-j}{j}\phi(j) \;\sim\; n\left(\Phi_0(n) - \Phi_0(\frac{n}{2})\right) - \left(\Phi_1(n) - \Phi_1(\frac{n}{2})\right).$$

Using the expressions $\Phi_0(n) \sim 3n^2/\pi^2$, and $\Phi_1(n) \sim 2n^3/\pi^2$ from Theorem 2, and adding the terms for $h_1(n)$ and $h_2(n)$, we obtain

$$h(n) \sim \left(\frac{1}{4} + 3 - \frac{3}{4} - 2 + \frac{1}{4}\right)\frac{n^3}{\pi^2} = \frac{3n^3}{4\pi^2} \sim \frac{3}{8}\Phi_1(n)$$

as claimed.

Theorem 5. *For D large, there exists a set of lattice points in the plane with diameter D and convex-hull $\mathcal{C}$ such that*

$$|\mathcal{C}| \sim 12\ \sqrt[3]{\frac{2}{9\pi^2}}\ D^{2/3} = 3.388..D^{2/3}.$$

Proof. If we set $a = b = \frac{2}{3\sqrt{2}}\ D$ where D is of the form $\Phi_1(n)$, and use the construction for $\mathcal{C}(n)$, then $r \sim D/2$ by (18). By Lemma 2

$$|\mathcal{C}(n)| \sim \frac{3}{\sqrt[3]{4\pi^2}}\ \left(\frac{4}{3\sqrt{2}}\right)^{2/3} D^{2/3} = 3\ \sqrt[3]{\frac{2}{9\pi^2}} D^{2/3}. \tag{19}$$

Since identical convex-hulls are constructed in all four quadrants to obtain $\mathcal{C}$ from $\mathcal{C}(n)$, the diameter of $\mathcal{C}$ is D and $|\mathcal{C}| \sim 4|\mathcal{C}(n)|$. This proves the claim of the theorem when combined with (19).

References

1. George E. Andrews. An asymptotic expression for the number of solutions of a general class of diophantine equations. *Transactions of the AMS* 99:272–277, May 1961.
2. B. Balog and I. Bárány. On the convex hull of integer points in a disc. *DIMACS Series, Discrete and Computational Geometry*, 6:39–44, 1991.
3. I. Bárány, R. Howe, and L. Lovasz. On integer points in polyhedra: a lower bound. *Combinatorica*, 12:135–142, 1992.
4. I. Bárány and David G. Larman. The convex hull of integer points in a large ball. *Math. Ann.*, 312:167–181, 1998.
5. W. Cook, M. Hartman, R. Kannan, and C. McDiarmid. On integer points in polyhedra. *Combinatorica*, 12:27–37, 1992.
6. A. Efrat and C. Gotsman. Subpixel image registration using circular fiducials. *Internat. J. Comput. Geom. Appl.*, 4(4):403–422, 1994.
7. S. Har-Peled. An output sensitive algorithm for discrete convex hulls. *Computational Geometry*, 10:125–138, 1998.
8. G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers.* Oxford University Press, April 1980.
9. M. D. Katz and D. J. Volper. Data structures for retrieval on square grids. *SIAM J. Comput.*, 15(4):919–931, November 1986.
10. A. M. Vershik. On the number of convex lattice polytopes. *Geometry and Functional Analysis*, 2:381–393, 1992.

On the Planar Two-Watchtower Problem*

Sergei Bespamyatnikh[1], Zhixiang Chen[2], Kanliang Wang[3], and Binhai Zhu[4]

[1] Department of Computer Science, University of British Columbia, Vancouver, B.C. V6T 1Z4, Canada.
besp@cs.ubc.ca.
[2] Department of Computer Science, University of Texas at Pan American, Edinburg, TX 78539-2999, USA.
chen@cs.panam.edu.
[3] School of Management, Xi'an Jiaotong University, Xi'an, Shaanxi, China.
klwang@xjtu.edu.cn.
[4] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA.
bhz@cs.montana.edu.

Abstract. In this paper we study the following planar two-watchtower problem: given a terrain (X-monotone chain) T with n vertices, locate two vertical segments (*watchtowers*) $\overline{uv}$ and $\overline{u'v'}$ which have to be erected on T such that every point on T is visible to the top of either watchtowers (u or u') and the maximum height of $\overline{uv}, \overline{u'v'}$ is minimized. We present an $O(n^4)$ time algorithm to solve the discrete version of this problem when v, v' must be on some vertices of T. Under a mild condition on solving a special cubic equation with three bounded variables in $O(f_3)$ time we can also generalize the algorithm to solve the general problem in $O(n^4 + n^3 f_3)$ time. Using parametric search, the discrete problem can be solved in $O(n^3 \log^2 n)$ time and the general problem can be solved in $O(n^4 \log^2 n)$ time.

1 Introduction

Given a polyhedral terrain (mountain landscape), the shortest watchtower problem is to compute the shortest vertical segment $\overline{uv}$ erected on the terrain such that every point on the terrain can be seen from the top of $\overline{uv}$, say u. The problem arises in geographical information systems, military surveillance, etc. The problem was first studied by Sharir [Sh88], who presented an $O(n \log^2 n)$ time algorithm (where n is the size of the terrain). The algorithm was later improved by Zhu to $O(n \log n)$ time [Zh97]. On the related problems of guarding polyhedral terrains, readers are referred to [BDEG94,BSTZ97,O'R97,WZ00,Zh94].

A very natural generalization of the problem is to build a set of watchtowers which can collectively guard the terrain. The objective is certainly to minimize

* This research is supported by Research Grants Council of Hong Kong SAR, China (CERG Project No. CityU1103/99E) and a MONTS grant.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 121–130, 2001.
© Springer-Verlag Berlin Heidelberg 2001

the maximum (or sum) of the heights of the watchtowers. This general problem (in 3D) is NP-hard as the problem of deciding whether a set of points (which can be thought of as watchtowers with zero height) can collectively guard a given terrain is NP-hard [CS89]. Eidenbenz *et al.* proved that positioning the minimum number of watchtowers with fixed height (to cover a 3D terrain) is not only NP-hard, but also NP-hard to approximate within a ratio of $C \log n$, where C is a constant [ESW98]. However, if the number of towers $k (k > 1)$ is fixed then almost nothing is known, yet the problem seems to be difficult even for $k = 2$ in 3D. In this paper we first study the corresponding problem in 2D, i.e., given a terrain in 2D, compute two watchtowers $\overline{u_1v_1}$ and $\overline{u_2v_2}$ (with heights $|u_1v_1|$ and $|u_2v_2|$ respectively) such that the terrain can be collectively guarded by the two watchtowers and the maximum of $\{|u_1v_1|, |u_2v_2|\}$ is minimized.

Regarding guarding a 2D terrain (monotone chain), Chen *et al.* proved that it is NP-hard to compute the minimum number of guards (which can only lie on the 2D terrain) to cover the whole terrain [CEU95]. If the guards are all at fixed height above the 2D terrain, then Nilsson's result implies a polynomial time solution to find the minimum number of guards [Ni94]. Our problem (even in 2D) is new and never been studied before, to the best of our knowledge.

We first present some basic definitions related to this paper. Given a point p in the plane, let $x(p), y(p)$ be the x, y-coordinate of p respectively. A polygonal chain $< v_1, ..., v_n >$ is said to be *X-monotone* if $x(v_i) > x(v_j)$ implies $i > j$, for all i and j between 1 and n. A *terrain* T in 2D is a X-monotone chain with vertices $v_1, ..., v_n$. Given two points a, b on or above T, a, b are *visible* to each other (or a, b *covers* each other) if the segment $\overline{ab}$ does not intersect any edge of T. The *visibility region* of $e \in T$ from a, is the set of all points of $e \in T$ which are visible to a. These definitions can all be found in [PS85,O'R87,O'R94].

2 Algorithm

In this section, we present implementable polynomial time algorithms for the planar 2-watchtower problem. We also consider the more theoretical solutions using parametric search. We first consider a *discrete* version of the problem, namely, when the base (bottom vertex) of each of the two watchtowers must be at a vertex of T. The following property regarding 2D visibility is easy to prove.

Lemma 1. *Given any point u above T, the non-empty visibility region of any edge $\overline{v_iv_{i+1}}$ from u is connected; moreover, if $x(u) < x(v_i)$ then the non-empty visibility region contains v_{i+1} and if $x(u) > x(v_{i+1})$ then the non-empty visibility region contains v_i.*

2.1 Algorithm for the Discrete Planar 2-Watchtower Problem

We begin with some definitions. Given a point p on T, we call the minimum height watchtower located at v_i whose top can see p the *critical watchtower* of p and v_i. We say that the height of this critical watchtower is the *critical height*

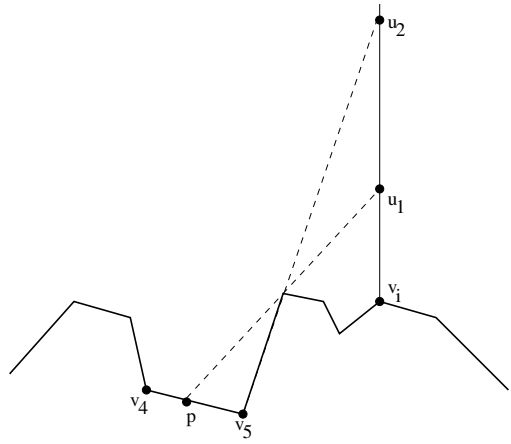


Fig. 1. The definition of critical watchtower.

of p and v_i and denote it by $h(p, v_i)$. Clearly $h(p, v_i)$ is the minimum height of a watchtower one can build which is located at v_i and its top can see p. In Figure 1, $\overline{u_1 v_i}$ is the critical watchtower of p and v_i and its height $|u_1 v_i|$ is exactly $h(p, v_i)$. Similarly we define the *critical watchtower* of an edge e of T and v_i as the minimum height watchtower located at v_i whose top can see every point of e and we denote the *critical height* of this tower by $h(e, v_i)$. In Figure 1, $\overline{u_2 v_i}$ is the critical watchtower of $e_4 = \overline{v_4 v_5}$ and v_i, and $h(e_4, v_i) = |u_2 v_i|$. (Notice that the concept of critical watchtower can be easily generalized to arbitrary point v on T.) In the following we first prove some fundamental properties (a necessary condition) of the *discrete* planar 2-watchtower problem.

Lemma 2. *When i, j are fixed, if $\overline{u_i v_i}$ ($\overline{u_j v_j}$) achieves the optimal solution for the discrete 2-watchtower problem then u_i, u_j collectively see T and there exists a point q on T such that $h(q, v_j) \geq h(q, v_i)$ ($h(q, v_i) \geq h(q, v_j)$).*

Based on this lemma, we have the following algorithm to solve the discrete 2-watchtower problem. Without loss of generality, we assume that i, j are fixed for now. We have to consider two cases: (1) q is a vertex of T, $q \neq v_1, q \neq v_n$, and (2) q is a point on some edge of S. Notice that in case 1, following Lemma 1, one of the edges incident to q must be completely seen by the optimal watchtower $\overline{u_i v_i}$ ($\overline{u_j v_j}$). Moreover, if $q = v_d$, $d < i$, then the edge $\overline{v_{d-1} v_d}$ must be completely seen by u_i. Symmetrically, if $q = v_d$, $d > i$, then the edge $\overline{v_d v_{d+1}}$ must be completely seen by u_i. Therefore, to handle case 1 we do the following: for each edge e we compute $h(e, v_i)$ and $h(e, v_j)$, if $h(e, v_j) \geq h(e, v_i)$ we build two towers at v_i, v_j with height $h(e, v_i)$, otherwise we build two towers at v_i, v_j with height $h(e, v_j)$. Then we check whether the tops of the two towers, u_i and u_j, can collectively see T. For a fixed e, this can be done in linear time: just compute the visibility region of T from u_i, u_j and then check whether every edge can be collectively covered by u_i and u_j. This can be done in linear time using the standard visibility

computation subroutines [EA81,Le83,Jo90,JS87,O'R87]. We record all pairs of the towers when their tops can collectively cover T. Among them we pick up the pair with the minimum height as $H_1(T, i, j)$. Clearly $H_1(T, i, j)$ can be computed in $O(n^2)$ time and $O(n)$ space as there are $O(n)$ edges in T. We remark that $h(e, v_i)$ over all e can be pre-computed in linear time with the shortest path map rooted at v_i [GHLST87], but that will not affect the overall complexity for computing $H_1(T, i, j)$.

For case 2, where a point in the middle of some edge determines the optimal solution for the discrete two-watchtower problem, we do the following. We fix an edge e and assume that q must be on e (eventually we need to try on all such e). We first prove the following lemma.

Lemma 3. *When i, j are fixed, in case 2, q cannot be to the left (right) of both v_i and v_j. In other words, e must be between v_i and v_j.*

Now we show how to handle case 2. Without loss of generality, we assume $x(v_i) < x(v_j)$ and $e = \overline{v_k v_{k+1}}$ is between v_i and v_j. We first compute the shortest paths from v_k to v_i and from v_{k+1} to v_j. Let the two paths be $< p_1 = v_k, p_2, ..., p_h = v_i >$ and $< q_1 = v_{k+1}, q_2, ..., q_m = v_j >$ respectively (Figure 2).

Notice that if we extend the edges $\overline{p_s p_{s+1}} (s = 1, ..., h-2)$, they naturally divide e into $O(h)$ red intervals and at the same time the intersections of these extensions with the vertical line through v_i determines the heights of the towers located at v_i. We can define a function $F_r(q)$ for each point q on e such that it is the height of the tower located at v_i and the top of the tower can see q, but nothing to the left of it. Clearly $F_r(q)$ is piecewise decreasing function (containing $O(h)$ pieces). Symmetrically, we can extend the edges $\overline{q_t q_{t+1}} (t = 1, ..., m-2)$, they also naturally divide e into $O(m)$ blue intervals. We can also define a function $F_b(q)$ for each point q on e such that it is the height of the tower located

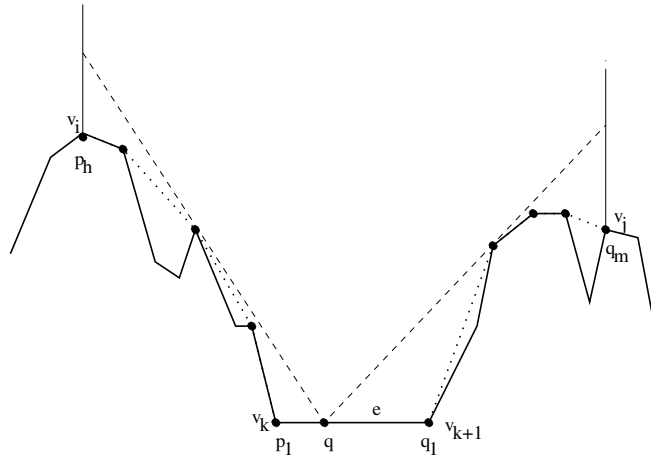


Fig. 2. Handling case 2 for the discrete two-watchtower problem.

at v_j and the top of the tower located at v_j can see q, but nothing to the right of it. Again $F_b(q)$ is a piecewise increasing function (containing $O(m)$ pieces). (*In other words, $F_r(q), F_b(q)$ are exactly $h(q, v_i), h(q, v_j)$ respectively.*) Let (X, Y) be the coordinate of q, (x, y) be the coordinate of u_i and let $p_s = (x(p_s), y(p_s))$ be the vertex of T tangent to $\overline{u_i q}$. The height of $\overline{u_i v_i}$ is the difference between y and $y(v_i)$, which is

$$(x(v_i) - x(p_s))\frac{Y - y(p_s)}{X - x(p_s)} + y(p_s) - y(v_i).$$

As Y is a linear function of X, $F_r(q)$ (and $F_b(q)$) is of the form

$$a_1\frac{X - a_2}{X - a_3} + a_4,$$

where X represents the x-coordinate of q which is bounded by $x(v_k)$ and $x(v_{k+1})$, and a_1, a_2, a_3, a_4 are constants related to the local geometry.

Now the problem is to find the intersection of the two functions $F_r()$ and $F_b()$ at some optimal point $q^*(e, i, j)$ (with respect to e). (Geometrically, $q^*(e, i, j)$ is the point such that if one moves it by any small distance along e, the maximum height of the two towers whose top must both see $q^*(e, i, j)$ will be increased.) What we do is to first merge the red/blue interval points into a sorted list (which can be done in $O(h + m)$ time), then we use prune and search to locate the interval which contains the intersection of $F_r()$ and $F_b()$. This can be done in $O(h + m) = O(n)$ time: we first choose the $\lfloor \frac{h+m}{2} \rfloor$-th interval point q', if $F_r(q') > F_b(q')$ then we throw away all the interval points to the left of q'; if $F_r(q') < F_b(q')$ then we throw away all the interval points to the right of q'; if $F_r(q') = F_b(q')$ then we are done. This procedure continues until either we have $F_r(q') = F_b(q')$ (in this case $q' = q^*(e, i, j)$) or we have only two sub-intervals left, one red and one blue. At this stage we simply solve an equation $F_r(q) = F_b(q)$ to obtain $q^*(e, i, j)$, which can be done in $O(1)$ time as this is a 1-variable quadratic equation. To evaluate $F_r(q')$ and $F_b(q')$ in $O(1)$ time, we just keep track of, for each interval point q' on e, the previous and successive red and blue interval points. This can be easily done when we merge the red and blue interval points. Therefore, the time it takes to compute the optimal solution when both i, j and e are fixed is

$$O(n) + O(n/2) + O(n/4) + ... + O(1) = O(n).$$

It remains to show that the tops of the two towers can collectively cover T. This can be done in a similar fashion as in case 1. We just need to compute the visibility region of (every edge of) T from u_i, u_j and then check whether each edge is collectively covered by u_i and u_j. This can be done in $O(n)$ time. If u_i, u_j cannot collectively cover T, we simply throw them away as valid candidates. As we might have $O(n)$ edges between v_i, v_j we have to try all of them and record all the pair of towers whose tops collectively cover T. Among those recorded pairs we pick up the one whose height is the shortest as $H_2(T, i, j)$. Therefore we have the following lemma.

Lemma 4. *When i, j are fixed, the optimal solution for the discrete 2-watchtower problem can be obtained in $O(n^2)$ time and $O(n)$ space.*

With Lemma 4, it is straightforward to have the following theorem as we have $O(n^2)$ number of combinations of i, j.

Theorem 1. *The optimal solution for the discrete 2-watchtower problem can be obtained in $O(n^4)$ time and $O(n)$ space.*

Finally, we show that with parametric search [Me83] the discrete 2-watchtower problem can be solved in $O(n^3 \log^2 n)$ time. The height h of two towers is a parameter of the decision problem which asks whether there exist a pair of vertices v_i and v_j of the terrain T such that every point of the terrain is visible from towers of height h located at v_i and v_j. We need two algorithms, sequential and parallel, for the decision problem. Let T_s be the running time of the sequential decision algorithm. (From the previous discussion, it is easy to see $T_s = O(n^3)$.) Let T_p and P denote the running time and the number of processors of the parallel algorithm respectively. The parametric searching scheme allows us to solve the optimization problem, i.e., the planar discrete 2-watchtower problem, in $O(PT_p + T_P T_s \log P)$ time. We use a parallel algorithm of Atallah et al. [ACW91] for computing visibility in a simple polygon. Their algorithm is based on a divide-and-conquer technique and can be implemented in the weak parallel model of computation of Valiant [Va75]. The running time is $O(\log n)$ using $O(n/\log n)$ processors. We just need a simplified version of the algorithm with $O(\log n)$ running time and $O(n)$ processors. We apply the algorithm in parallel for all pairs of v_i, v_j using $O(n^3)$ processors. Hence $T_p = O(\log n)$ and $P = O(n^3)$.

Theorem 2. *With parametric search, the optimal solution for the discrete 2-watchtower problem can be obtained in $O(n^3 \log^2 n)$ time.*

Notice that so far it is not possible to actually implement an algorithm using parametric search. Therefore, this result is only of theoretical meaning.

2.2 Algorithm for the Planar 2-Watchtower Problem

We now discuss the solutions for the planar 2-watchtower problem. Our idea is basically the same as that for the discrete case. Again the following lemma, which presents a necessary condition for the planar two-watchtower problem, is easy to prove.

Lemma 5. *If $\overline{uv}$ ($\overline{u'v'}$) achieves the optimal solution for the planar 2-watchtower problem then u, u' collectively see T and there exists a point q on S such that $h(q, v') \geq h(q, v)$ ($h(q, v) \geq h(q, v')$).*

Similar to the discrete case, we also have two cases when q is a vertex of T and q is on some edge of T. However, for both cases v and v' do not necessarily locate on a vertex of T anymore. Let L be the set of lines which are the extensions of edges of T. We compute the arrangement of L, $A(L)$. We have the following lemma.

Lemma 6. *In case 1, an optimal solution for the planar 2-watchtower* $\overline{uv}$ *has the property that either* v *is a vertex of* T *or* u *is a vertex of* $A(L)$*.*

Proof. It suffices to prove the second part of the lemma. Assume that $\overline{u^*v^*}$ is the optimal solution for the planar 2-watchtower problem of T such that v^* is not a vertex of T and u^* is not a vertex of $A(L)$. We show below how to construct $\overline{uv}$ such that every point of T which is covered only by u^* is also covered by u and $|uv| \leq |u^*v^*|$.

By the definition of the problem, under case 1, u^* must be on a line l^* of L and v^* is on an edge e of T (Figure 3). If l^* and e are parallel then we can slide $\overline{u^*v^*}$ to either a vertex of e or the closest vertex u^+ of $A(L)$ on l^*, which is defined by l^* and l^+ such that some points on T seen by u^+ are not seen by the top of the other watchtower. Let the resulting watchtower be $\overline{uv}$, then $|uv| = |u^*v^*|$ and either v is a vertex of T or u is a vertex of the arrangement $A(L)$.

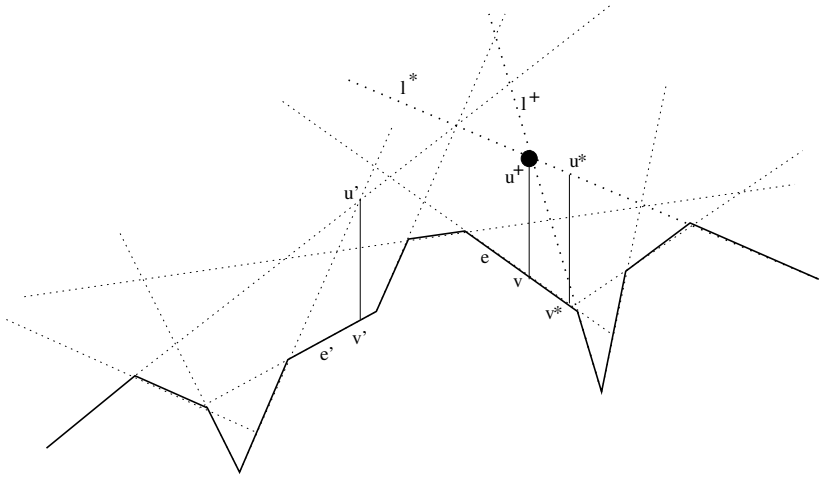


Fig. 3. Illustration for Lemma 6.

If l^* and e are not parallel then we can slide $\overline{u^*v^*}$ (this time assume that we can always squeeze it) to either a vertex of e or the closest vertex u^+ of $A(L)$ on l^*, which is again defined by l^* and l^+ such that some points on T seen by u^+ are not seen by the top of the other watchtower and the height $|u^*v^*|$ becomes shorter in the process of sliding. Let the resulting watchtower be $\overline{uv}$, then $|uv| < |u^*v^*|$ and either v is a vertex of T or u is a vertex of the arrangement $A(L)$. □

The proof of Lemma 6 naturally gives us an algorithm to solve the problem under case 1 in $O(n^4)$ time, by converting the solutions of the $O(n^2)$ discrete 2-watchtower problems, where the bases of the towers are on some vertices of T, to the general problem. We thus have the following lemma.

Lemma 7. *In case 1, the optimal solution for the planar 2-watchtower problem can be obtained in $O(n^4)$ time and $O(n^2)$ space.*

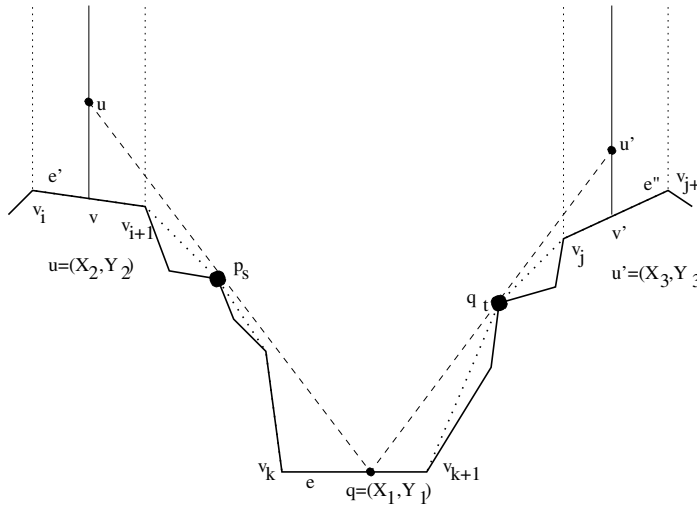


Fig. 4. Handling case 2 for the general two-watchtower problem.

Case 2 is similar to the one we handle in the previous subsection. The difference is that now v, v' can be on some edges of T. Let $v \in e', v' \in e", q \in e$ and without loss of generality let $e' = \overline{v_i v_{i+1}}, e" = \overline{v_j v_{j+1}}$ and $e = \overline{v_k v_{k+1}}$ (Figure 4). Now the function which defines the critical height $h(q, v)$, $F_r(e', q)$, where $q \in e$ and $v \in e'$, is a piecewise decreasing function when v is fixed. Let $q = (X_1, Y_1)$ and $u = (X_2, Y_2)$. $F_r(e, q)$ is of the form

$$a_1(X_2 - a_2)\frac{X_1 - a_3}{X_1 - a_4} + a_5 X_2 + a_6, b_1 \leq X_1 \leq b_2, b_3 \leq X_2 \leq b_4, - - -(A)$$

where $a_i (1 \leq i \leq 6), b_j (1 \leq j \leq 4)$ are constants related to local geometry. Similarly, the function $F_b(e", q)$, where $q \in e$ and $v' \in e"$, defines the critical height $h(q, v')$. It is a piecewise increasing function when v' is fixed and of the same form as (A). Notice that once X_1 is fixed we can easily maximize and minimize the value of (A) in $O(1)$ time. This evaluation will be used in the prune and search step. Similar to the previous section, the prune and search step stops when we have obtained an optimal interval point or we have located a red and a blue intervals such that $F_r(e, q), F_b(e, q)$ overlap (i.e., within the intersection of these two intervals the maximum of $F_r(e, q)$ is larger than the minimum of $F_b(e, q)$ and vice versa).

Finally, if we assume that the following equation with variables X_1, X_2, X_3,

$$a_1(X_2 - a_2)\frac{X_1 - a_3}{X_1 - a_4} + a_5 X_2 + a_6 = a'_1(X_3 - a'_2)\frac{X_1 - a'_3}{X_1 - a'_4} + a'_5 X_3 + a'_6, - - -(B)$$

where $a_i, a_i'(1 \leq i \leq 6)$ are constants and $b_1 \leq X_1 \leq b_2, b_3 \leq X_2 \leq b_4, b_3' \leq X_3 \leq b_4'$ for constants $b_1, b_2, b_3, b_4, b_3', b_4'$, can be solved in f_3 time (which can be done numerically in practice), then the merging of red/blue list, the subsequent prune and search step and the checking of whether u, u' can collectively cover T will all be done in $O(n + f_3)$ time with $e, e', e"$ fixed. As we have $O(n)$ candidates for e and $O(n^2)$ number of combinations of $e', e"$, case 2 can be solved in $O(n^4 + n^3 f_3)$ time. We hence have the following lemma.

Lemma 8. *In case 2, if we assume that the above cubic equation (B) can be solved in f_3 time then the planar 2-watchtower problem can be solved in $O(n^4 + n^3 f_3)$ time and $O(n)$ space.*

Therefore we have the following theorem.

Theorem 3. *Assuming that the above cubic equation (B) can be solved in (approximately) f_3 time then the optimal solution for the planar 2-watchtower problem can be obtained in $O(n^4 + n^3 f_3)$ time and $O(n^2)$ space.*

Finally, we show that with parametric search the planar 2-watchtower problem can be solved in $O(n^4 \log^2 n)$ time. The height H of two towers is a parameter of the decision problem which asks whether there exist a pair of towers sliding on e' and $e"$ of the terrain T such that every point of the terrain is visible from towers of height H located on e' and $e"$. Let T_s be the running time of the sequential decision algorithm. It is easy to see that $T_s = O(n^4)$: for each triple $e, e', e"$, there are $O(n)$ event points (endpoints of the red/blue intervals) and checking whether the terrain T is completely visible can be done in $O(n)$ time. Let T_p and P denote the running time and the number of processors of the parallel algorithm respectively. It is easy to see that $T_p = O(\log n)$ and $P = O(n^4)$: we use a processor for each $e, e', e"$ and an event point on e (this uses a total of $O(n^4)$ processors); moreover, with these processors it is easy to check in parallel whether T can be completely covered in $O(\log n)$ time. Hence the total running time is $O(n^4 \log^2 n)$. (Again, this result is only of theoretical meaning.)

Theorem 4. *With parametric search, the optimal solution for the planar 2-watchtower problem can be obtained in $O(n^4 \log^2 n)$ time.*

3 Closing Remarks

In this paper, we present several polynomial time algorithms to solve the planar 2-watchtower problems. An immediate open problem is how to solve the 2-watchtower problem in 3D efficiently. (We remark that for the discrete 2-watchtower problem in 3D, where the towers must be located at the vertices of the terrain, it is possible to use a necessary condition similar to Lemma 2 to obtain an $O(n^6)$ time solution.)

References

[ACW91] M.J. Atallah, D.Z. Chen, and H. Wagner, An optimal parallel algorithm for the visibility of a simple polygon from a point, *J. ACM,* 38(3):516-533, 1991.

[BDEG94] M. Bern, D. Dobkin, D. Eppstein, and R. Grossman, Visibility with a moving point of view, *Algorithmica,* 11:360-378, 1994.

[BSTZ97] P. Bose, T. Shermer, G. Toussaint, B. Zhu, Guarding polyhedral terrains, *Comp. Geom. Theory and Appl.,* 7:173-186, 1997.

[CEU95] D.Z. Chen, V. Estivill-Castro and J. Urrutia, Optimal guarding of polygons and monotone chains, *Proc. 7th Canadian Conf. on Computational Geometry*, pages 133-138, 1995.

[CS89] R. Cole and M. Sharir, Visibility problems for polyhedral terrains, *J. Symbolic Computation,* 7:11-30, 1989.

[EA81] H. ElGindy and D. Avis, A linear time algorithm for computing the visibility polygon from a point, *J. Algorithms,* 2(4):186-197, 1981.

[ESW98] S. Eidenbenz, C. Stamm and P. Widmayer, Positioning guards at fixed height above a terrain — and optimum inapproximability result, *Proc. 6th European Symp. on Algorithms (ESA'98)*, pages 187-198, 1998.

[GHLST87] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica,* 2(2):209-234, 1987.

[Jo90] B. Joe, On the correction of a linear time visibility polygon algorithm, *Intl. J. Comput. Math.,* 32:155-172, 1990.

[JS87] B. Joe and R.B. Simpson, Correction to Lee's visibility polygon algorithm, *BIT,* 27:458-473, 1987.

[Le83] D.T. Lee, Visibility of a simple polygon, *Comput. Vision Graph. Image Process.,* 22:207-221, 1983.

[Me83] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM,* 30(4):852-965, 1983.

[Ni94] B. Nilsson, *Guarding Art Galleries - Methods for Mobile Guards*, Doctoral thesis, Department of Computer Science, Lund University, 1994.

[O'R87] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, 1987.

[O'R94] J. O'Rourke, *Computational geometry in C*, Cambridge University Press, 1994.

[O'R97] J. O'Rourke, Vertex π-lights for monotone mountains, *9th Canadian Conference on Computational Geometry*, Queen's University, August 11-14, 1997. http://www.dgp.toronto.edu/cccg/cccg97/papers/5/5.html

[PS85] F.P. Preparata, M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, 1985.

[Sh88] M. Sharir, The shortest watchtower and related problems for polyhedral terrains, *Information Processing Letters*, 29(5):265-270, 1988.

[Va75] L. Valiant, Parallelism in comparison problems, *SIAM J. Computing*, 4:348-355, 1975.

[WZ00] C.A. Wang and B. Zhu, Three dimensional weak visibility: complexity and applications, *Theoretical Computer Science*, 234(1/2):219-232, 2000.

[Zh97] B. Zhu, Computing the shortest watchtower of a polyhedral terrain in $O(n \log n)$ time, *Computational Geometry: Theory and Applications*, 8:181-193, 1997.

[Zh94] B. Zhu, *Computational Geometry in Two and a Half Dimensions*, PhD thesis, School of Computer Science, McGill University, 1994.

Efficient Generation of Triconnected Plane Triangulations

Shin-ichi Nakano

Gunma University, Kiryu 376-8515, Japan,
nakano@cs.gunma-u.ac.jp
Fax: +81-277-30-1812

Abstract. A "rooted" plane triangulation is a plane triangulation with one designated vertex on the outer face. In this paper we give a simple algorithm to generate all triconnected rooted plane triangulations with at most n vertices. The algorithm uses $O(n)$ space and generates such triangulations in $O(1)$ time per triangulation without duplications. The algorithm does not output entire triangulations but the difference from the previous triangulation. By modifying the algorithm we can generate all triconnected rooted plane triangulation having exactly n vertices including exactly r vertices on the outer face in $O(r)$ time per triangulation without duplications, while the previous best algorithm generates such triangulations in $O(n^2)$ time per triangulation. Also we can generate without duplications all triconnected (non-rooted) plane triangulations having exactly n vertices including exactly r vertices on the outer face in $O(r^2n)$ time per triangulation, and all maximal planar graphs in $O(n^3)$ time per graph.

1 Introduction

Generating all graphs with some property without duplications has many applications, including unbiased statistical analysis[M98]. A lot of algorithms to solve these problems are already known [A96, B80, LN01, M98, W86, etc]. See nice textbooks [G93, KS98]. In this paper we wish to generate all triconnected "rooted" plane triangulations, which will be defined precisely in Section 2, with at most n vertices. It is known that every triconnected planar graph has a unique embedding on a sphere only up to mirror copy[HW74].

To solve these all-graph-generating problems some types of algorithms are known. Classical method algorithms [G93, p57] first generate all the graphs with given property allowing duplications, but output only if the graph has not been output yet. Thus this method requires quite a huge space to store a list of graphs that have already been output. Furthermore, checking whether each graph has already been output requires a lot of time. Orderly method algorithms [G93, p57] need not to store the list, since they output a graph only if it is a "canonical" representative of each isomorphism class. Reverse search method algorithms [A96] also need not to store the list. The idea is to implicitly define a connected graph H such that the vertices of H correspond to the graphs with

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 131–141, 2001.
© Springer-Verlag Berlin Heidelberg 2001

the given property, and the edges of H correspond to some relation between the graphs. By traversing an implicitly defined spanning tree of H, one can find all the vertices of H, which correspond to all the graphs with the given property.

The main idea of our algorithm is that for some problems we can define a tree (not a general graph) as the graph H of reverse search method. Thus our algorithm does not need to find a spanning tree of H, since H itself is a tree. With some other ideas we give the following two simple but efficient algorithms.

Our first algorithm generates all triconnected rooted plane triangulations with at most n vertices. A *rooted* plane triangulation means a plane triangulation with one designated "root" vertex on the outer face. For instance there are four triconnected rooted plane triangulations with at most five vertices including exactly three vertices on the outer face, as shown in Fig. 1(a). The root vertices are depicted by white circles. However, there are only two triconnected plane triangulations with at most five vertices including exactly three vertices on the outer face. See Fig. 1(b). The algorithm uses $O(n)$ space and runs in $O(f(n))$ time, where $f(n)$ is the number of nonisomorphic triconnected rooted plane triangulations with at most n vertices. The algorithm generates triangulations without duplications. So the algorithm generates each triangulation in $O(1)$ time on average. The algorithm does not output entire triangulations but the difference from the previous triangulation.

By modifying our first algorithm we can generate without duplications all triconnected rooted plane triangulations having exactly n vertices including exactly r vertices on the outer face. The algorithm uses $O(n)$ space and runs in $O(r \cdot f(n, r))$ time, where $f(n, r)$ is the number of nonisomorphic such triangulations. So the algorithm generates each triangulation in $O(r)$ time on average, while the previous best algorithm [A96] generates such triangulations in $O(n^2)$ time per triangulation.

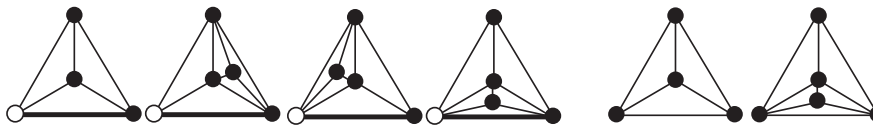


Fig. 1. (a) Triconnected rooted plane triangulations, and (b) triconnected plane triangulations.

Also we can generate all triconnected (non-rooted) plane triangulations having exactly n vertices including exactly r vertices on the outer face in $O(r^2 n)$ time (on average) per triangulation, and all maximal planar graphs in $O(n^3)$ time per graph. Our algorithm is simple and does not need the complicated theoretical linear-time plane-graph-isomorphism algorithm in [HW74].

For full version of the paper see [N01]. We also solved similar problems for biconnected cases [LN01].

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 shows a tree structure among triconnected rooted plane triangulations. Section 4 presents our first algorithm. By modifying the algorithm we give three more algorithms in Section 5.

2 Preliminaries

Let G be a connected graph with n vertices. An edge connecting vertices x and y is denoted by (x, y). The *degree* of a vertex v, denoted by $d(v)$, is the number of neighbors of v in G. A *cut* is a set of vertices whose removal results in a disconnected graph or a single-vertex graph K_1. The *connectivity* $\kappa(G)$ of a graph G is the cardinality of the minimum number of vertices consisting a cut. G is $k-$*connected* if $k \leq \kappa(G)$.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed planar embedding. A plane graph divides the plane into connected regions called *faces*. The unbounded face is called *the outer face*, and other faces are called *inner faces*. We regard *the contour* of a face as the clockwise cycle formed by the vertices and edges on the boundary of the face. We denote the contour of the outer face of plane graph G by $C_o(G)$. A vertex is *an outer vertex* of G if it is on $C_o(G)$, and *an inner vertex* otherwise. An edge is *an outer edge* of G if it is on $C_o(G)$, and *an inner edge* otherwise. A plane graph is called *a plane triangulation* if each inner face has exactly three edges on its contour. A *rooted* plane triangulation is a plane triangulation with one designated vertex on $C_o(G)$. The designated vertex is called *the root vertex*.

Let G be a triconnected plane triangulation. Assume G has a cut S consisting exactly three vertices, say x, y, z. Let $G^{'}$ be the graph derived from G by removing S. Let W be the vertex set of a connected componect of $G^{'}$. Let $G^{''}$ be the plane subgraph of G induced by $S \cup W$. Since G is triconnected each vertex in S has at least one neighbor in W. Since G is a plane triangulation, all vertices in S are on the some face, say F, of $G^{''}$. We now have the following lemma.

Lemma 1. *Let P be the path on the contour of F between x and y not containing z. Then the followings hold. (a) If x and y are inner vertices of G, then G has an (inner) edge (x, y). (b) If x is an inner vertex and y is an outer vertex of G, then G has an (inner) edge (x, y). (c) If x and y are outer vertices of G, then all vertices on P are also outer vertices.*

Thus if $G \neq K_4$, then at most two vertices of S are outer vertices, because by (c) above if all three vertices in S are outer vertices, then S cannot be a cut. If S has at most one outer vertex, then a cut $S = \{x, y, z\}$ of G is called *a separating triangle*, and by (a) and (b) above G has three edges (x, y), (y, z) and (z, x). If S has exactly two outer vertex, say x and y, and exactly one inner vertex, say z, then a cut $S = \{x, y, z\}$ of G is called *a separating fan* with *center* z, and by (b) above G has two edges (x, z), (y, z).

Then we define an operation on a plane triangulation. Assume (u, v) is an inner edge of a plane triangulation G. Let u, v, x and v, u, y be the two faces having (u, v) on its contour. By (i) deleting the edge (u, v), (ii) merging u and v into a new vertex w, and (iii) replacing edges (u, x) and (v, x) by (w, x), and edges (u, y) and (v, y) by (w, y), as shown in Fig. 2(a), we can have a new plane graph. This operation is called *a contraction* of inner edge (u, v).

Similarly, for an outer edge (u, v) of a plane triangulation G, let u, v, x be the face having (u, v) on its contour. By (i) deleting the edge (u, v), (ii) merging u and v into a new vertex w, and (iii) replacing edges (u, x) and (v, x) by (w, x), we can have a new plane graph. This operation is called *a contraction* of outer edge (u, v).

Let G be a triconnected plane triangulation. If G has a separating triangle $\{u, v, z\}$, then the contraction of (u, v) produces "parallel" edge (w, z), so the resulting plane graph is not a plane triangulation anymore. If G has a separating fan $\{u, v, z\}$ with center v, then after contracting (u, v), the resulting plane graph has a cut $\{w, z\}$, and is not triconnected anymore. If G has no cut consisting of u, v and exactly one more vertex, then the contraction of (u, v) produces again a triconnected plane triangulation. We say such edge (u, v) is *contractible*.

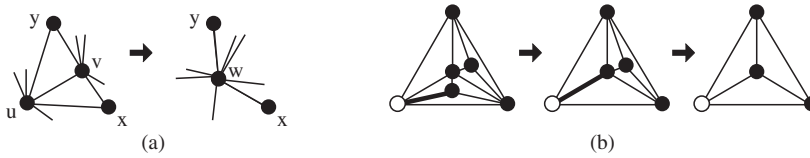


Fig. 2. (a) A contraction and (b) a contraction sequence.

3 The Contracting Sequence and the Genealogical Tree

Let S_n be the set of all triconnected rooted plane triangulations with at most n vertices. Here we explain a tree structure among the triangulations in S_n.

Let G be a triangulation in S_n but not K_4. Let $C_o(G) = v_1, v_2, \cdots, v_k$, and v_1 be the root vertex of G. A contractible edge (v_1, w_i) is *the first contractible edge* of G if none of $(v_1, w_2), (v_1, w_3), \cdots, (v_1, w_{i-1})$ is contractible edge, where $(v_1, v_k) = (v_1, w_1), (v_1, w_2), \cdots, (v_1, w_{d(v_1)}) = (v_1, v_2)$ are the edges incident to v_1, and assume that they appear around v_1 counterclockwise in this order.

Lemma 2. *Every triconnected rooted plane triangulation except K_4 has the first contractible edge.*

For each triconnected rooted plane triangulation G in S_n except K_4, contracting the first contractible edge from G results in also a triconnected rooted plane triangulation, denoted by $P(G)$, in S_n with one less vertices. Thus we can define the triangulation $P(G)$ in S_n for each G in S_n except K_4. We say G is a *child* triangulation of $P(G)$.

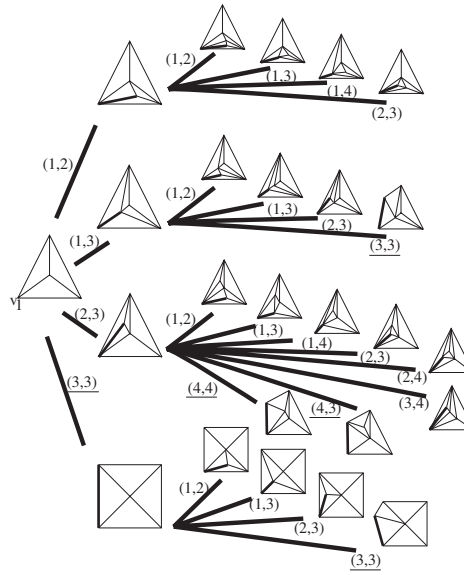


Fig. 3. Genealogical tree T_6.

Given a triangulation G in S_n, by repeatedly contracting the first contractible edge, we can have the unique sequence $G, P(G), P(P(G)), \cdots$ of triangulations in S_n which eventually ends with K_4. See an example in Fig. 2(b), in which the first contractible edges are depicted by thick lines.

By merging those sequences we can have *the genealogical tree* T_n of S_n such that the vertices of T_n correspond to the triangulations in S_n, and each edge corresponds to each relation between some G and $P(G)$. For instance T_6 is shown in Fig. 3, in which the first contractible edges are depicted by thick lines. We call the vertex in T_n corresponding to K_4 *the root* of T_n.

4 Algorithm

Given S_n we can construct T_n by the definition, possibly with huge space and much running time. However, how can we construct T_n efficiently only given an integer n? Our idea is by reversing the contracting procedure as follows.

Given a triconnected rooted plane triangulation G in S_n with at most $n-1$ vertices, we wish to find all child triangulations of G. Let $C_o(G) = v_1, v_2, \cdots, v_k$, and v_1 be the root vertex of G. Let $(v_1, v_k) = (v_1, w_1), (v_1, w_2), \cdots, (v_1, w_{d(v_1)}) = (v_1, v_2)$ be the edges incident to v_1, and assume that they appear around v_1 counterclockwise in this order. Assume (v_1, w_s) be the first contractible edge of G. Since especially only K_4 has no contractible edge, for convenience, we regard $(v_1, w_s) = (v_1, v_2)$ as the first contractible edge for K_4.

We denote by $G_i(i, j)$, where $1 \leq i < j \leq d(v_1)$, the rooted plane triangulation obtained from G by (1) removing edges $(v_1, w_{i+1}), (v_1, w_{i+2}), \cdots, (v_1, w_{j-1})$

(thus if $i+1=j$ we need not remove any edge), (2) adding new vertex v on the inner face of G containing v_1, w_i, w_j, (3) adding an edge (v_1, v) in the face, and (4) adding $j-i+1 \geq 2$ edges $(v, w_i), (v, w_{i+1}), \cdots, (v, w_j)$, as shown in Fig. 4(a). $G_i(i,j)$, where $1 \leq i < j \leq d(v_1)$, is a child triangulation of G if and only if (v_1, v) is the first contractible edge of $G_i(i,j)$. (This corresponds to a contraction of an inner edge. See Fig. 3.)

Also, we denote by $G_o(i,j)$, where $i = d(v_1)$ and $3 \leq j \leq d(v_1)$, the rooted plane triangulation obtained from G by (1) removing edges (v_1, w_j), (v_1, w_{j+1}), $\cdots$, $(v_1, w_{d(v_1)})$, (2) adding new vertex v on the outer face of G, (3) adding edges (v_1, v) and (v, v_2), and (4) adding edges $(v, w_{j-1}), (v, w_j), \cdots, (v, w_{d(v_1)})$, as shown in Fig. 4(b). $G_o(i,j)$, where $i = d(v_1)$ and $3 \leq j \leq d(v_1)$, is a child triangulation of G if and only if (v_1, v) is the first contractible edge of $G_o(i,j)$. (This corresponds to a contraction of an outer edge. See Fig. 3.)

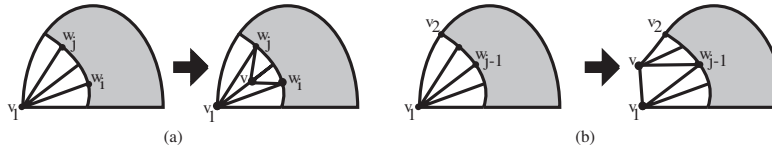


Fig. 4. Illustration for (a) $G_i(i,j)$ and (b) $G_o(i,j)$.

Thus $G_i(i,j)$ is a child triangulation of G having an inner edge as the first contractible edge, while $G_o(i,j)$ is a child triangulation of G having an outer edge as the first contractible edge. And each child triangulation is either $G_i(i,j)$ or $G_o(i,j)$ for some i and j.

For each t, $2 \leq t \leq d(v_1)-1$, we denote by $q(t)$ the largest index such that $(w_t, w_{q(t)})$ is an edge (chord) of G if such edge exists, and $q(t) = t$ otherwise. Now we check whether each $G_i(i,j)$ is a child triangulation of G, and check also for each $G_o(i,j)$, as follows. Let (v_1, w_s) be the first contractible edge of G.

Case 1: $G_i(i,j)$, where $1 \leq i < j \leq d(v_1)$. We have four subcases.

Case 1(a): $j \leq s$.

In this case (v_1, v) is the first contractible edge of $G_i(i,j)$, and $P(G_i(i,j)) = G$.

Case 1(b): $i < s < j$.

If G has a separating fan containing v_1 and w_i, then (v_1, v) is the first contractible edge for each j, $s < j \leq d(v_1)$ and $P(G_i(i,j)) = G$. Otherwise, if $j \leq q(i)$ then (v_1, v) is the first contractible edge and $P(G_i(i,j)) = G$. Otherwise (v_1, w_i) is a contractible edge of $G(i,j)$, and so $P(G_i(i,j)) \neq G$.

Case 1(c): $i = s$.

If $j = i+1$, then (v_1, v) is the first contractible edge of $G_i(i,j)$ (since v_1, w_s, w_{s+1} become a separating triangle) and $P(G_i(i,j)) = G$. Otherwise, $j \geq i+2$ holds then (v_1, w_s) is still contractible in $G_i(i,j)$, and $P(G_i(i,j)) \neq G$.

Case 1(d): $i > s$.

(v_1, v) is not the first contractible edge of $G_i(i,j)$. Thus $P(G_i(i,j)) \neq G$.

Case 2: $G_o(i,j)$, where $i = d(v_1)$ and $3 \leq j \leq d(v_i)$. We have three subcases.
Case 2(a): $s = d(v_1)$.
(Now each $w_2, w_3, \cdots, w_{d(v_1)-1}$ is contained in some separating fan.) For each $G_o(i,j)$, $3 \leq j \leq d(v_1)$, (v_1, v) is the first contractible edge, and $P(G_o(i,j)) = G$.
Case 2(b): $s = d(v_1) - 1$.
For each $G(i,j)$, $3 \leq j \leq d(v_1)$, (v_1, v) is the first contractible edge, and $P(G_o(i,j)) = G$. Note that in $G_o(i, d(v_1))$, $v_1, w_{d(v_1)-1}, v_2$ become a separating fan, and other cases are trivial.
Case 2(c): $s < d(v_1) - 1$.
If $j - 1 \geq s$ then (v_1, w_s) is still contractible in $G_o(i,j)$, thus (v_1, v) is not the first contractible edge and $P(G_o(i,j)) \neq G$. Otherwise if G has no separating fan containing v_1 and w_{j-1} then (v_1, w_{j-1}) is contractible in $G_o(i,j)$, thus (v_1, v) is not the first contractible edge, and $P(G_o(i,j)) \neq G$. Otherwise, for each j, $3 \leq j \leq s$, (v_1, v) is the first contractible edge in $G_o(i,j)$, and $P(G_o(i,j)) = G$.

Based on the case analysis above we can find all child triangulations of given triangulation in S_n. If G has k child triangulations, then we can find them in $O(k)$ time. This is an intuitive reason why our algorithm generates triangulations in $O(1)$ time per triangulation.

And recursively repeating this process from the root of T_n corresponding to K_4 we can traverse T_n without constructing whole T_n. During the traversal of T_n, we assign a label (i,j) to each edge connecting G and either $G_i(i,j)$ or $G_o(i,j)$ in T_n, as shown in Fig. 3. Each label denotes how to generate a child triangulation of G, and each sequence of labels on a path starting from the root specifies a triangulation in S_n. For instance $(1,2),(1,2)$ specify the uppermost triangulation in Fig. 3. During our algorithm we will maintain these labels only on the path from the root to the "current" vertex, because those are enough information to generate the "current" triangulation. To generate next triangulation, we need to maintain some more information (the first contractible edge (v_1, w_s), and $w_{q(t)}$ for each $1 < t < s$, etc.) only for the triangulations on the "current" path, which has length at most n. This is an intuitive reason why our algorithm uses only $O(n)$ space, while the number of triangulations may not be bounded by a polynomial in n.

```
   Procedure find-all-child-triangulations(G)
   begin
 1   output G { Output the difference from the previous triangulation}
 2   if G has exactly n vertices then return
 3   for i = 1 to s − 1
 4     for j = i + 1 to s
 5       find-all-child-triangulations(G_i(i, j))          { Case 1(a)}
 6   for i = 1 to s − 1
 7     if G has a separating fan containing v_1 and w_i then
 8       for j = s + 1 to d(v_1)
 9         find-all-child-triangulations(G_i(i, j))        { Case 1(b)}
10     else then
```

```
11        for j = s + 1 to q(i)
12          find-all-child-triangulations(G_i(i, j))        { Case 1(b)}
13    find-all-child-triangulations(G_i(s, s + 1))          { Case 1(c)}
14    if s = d(v_1) or d(v_1) − 1 then
15      for j = 3 to d(v_1)
16        find-all-child-triangulations(G_o(d(v_1), j))     { Case 2(a) and (b)}
17    else then
18      for j = 3 to s
19        if G has a separating fan containing v_1 and w_{j−1} then
20          find-all-child-triangulations(G_o(d(v_1), j))   { Case 2(c)}
    end

    Algorithm find-all-triangulations(n)
    begin
 1    output K_4
 2    if  n = 4 then return
 3    G = K_4
 4    find-all-child-triangulations(G_i(1, 2))
 5    find-all-child-triangulations(G_i(1, 3))
 6    find-all-child-triangulations(G_i(2, 3))
 7    find-all-child-triangulations(G_o(3, 3))
    end
```

Theorem 1. *The algorithm uses $O(n)$ space and runs in $O(f(n))$ time, where $f(n)$ is the number of nonisomorphic triconnected rooted plane triangulations with at most n vertices.*

5 Modification of the Algorithm

Then we consider our second problem. Let $S_{=n}^{=r}$ be the set of triconnected rooted plane triangulations having exactly n vertices including exactly r outer vertices. We wish to generate all triangulations in $S_{=n}^{=r}$ without duplications. Clearly all such triangulations are in T_n but with other triangulations. How can we output only triangulations in $S_{=n}^{=r}$, furthermore efficiently? Our idea is by pruning T_n, in the following way.

For each triangulation G in S_n but K_4, if the first contractible edge is an inner edge then $P(G)$ is a triangulation in S_n having the same number of outer vertices and one less inner vertices, otherwise the first contractible edge is an outer edge and $P(G)$ is a triangulation in S_n having one less outer vertices and the same number of inner vertices. Thus if G in S_n has exactly $n^{'}$ vertices including exactly $r^{'}$ outer vertices, then every "descendant" triangulation of G in T_n can have at most $r^{'} + (n - n^{'})$ outer vertices. Thus if $r > r^{'} + (n - n^{'})$ then none of descendant triangulations of G is in $S_{=n}^{=r}$. Also if G has $r^{'} > r$ outer vertices then every descendant triangulation of G has $r^{'}$ or more outer vertices, and so never in $S_{=n}^{=r}$. We prune such triangulations from T_n.

We prune T_n further based on the following lemma. We need some definitions here. Let S_n^i be the set of triconnected rooted plane triangulations with $n' < n$ vertices including exactly $r' = r$ outer vertices (so it already has enough outer vertices), and S_n^o be the set of triconnected rooted plane triangulations with $n' < n$ vertices including exactly $r' = r - (n - n')$ outer vertices (so it already has enough inner vertices). Let S_n^r be the set of triconnected rooted plane triangulations with $n' < n$ vertices including r', $r-(n-n') < r' < r$ outer vertices. Let G be a triangulation in S_n, $C_o(G) = v_1, v_2, \cdots, v_k$, and v_1 be the root vertex of G, $(v_1, v_k) = (v_1, w_1), (v_1, w_2), \cdots, (v_1, w_{d(v_1)}) = (v_1, v_2)$ be the edges incident to v_1, and assume that they appear around v_1 counterclockwise in this order. We call w_2 the *pivot* of G. If G has a separating fan containing both the root and the pivot of G, then it is called *the rp-fan* of G.

Lemma 3. *Let G be a triangulation in S_n with $n' < n$ vertices including r' outer vertices.*

(a) If G has the rp-fan, then (a1) if G has a child triangulation $G_i(1, j)$ for some j, then $G_i(1, j)$ has no rp-fan, (a2) if G has a child triangulation $G_i(2, j)$ for some j, then $G_i(2, j)$ has the rp-fan, (a3) if G has a child triangulation $G_i(i, j)$ for some $i \geq 3$ and j, then $G_i(i, j)$ has the rp-fan, and (a4) if G has a child triangulation $G_o(i, j)$ for some i and j, then $G_o(i, j)$ has the rp-fan.

(b) If G has no rp-fan, then (b1) if G has a child triangulation $G_i(1, j)$ for some j, then $G_i(1, j)$ has no rp-fan, (b2) G has no child triangulation $G_i(i, j)$ for any $i \geq 2$ and j, and (b3) G has no child triangulation $G_o(i, j)$ for any i and j.

Lemma 4. *Let G be a triangulation in S_n with $n' < n$ vertices including r', $r - (n - n') \leq r' < r$ outer vertices. If G has no rp-fan then G has no descendant triangulation in $S_{=n}^{=r}$. If G has the rp-fan then G has at least one descendant triangulation in $S_{=n}^{=r}$.*

Proof. By repeatedly applying Lemma 3(b), if G has no rp-fan then each descendant triangulation of G has no rp-fan and having exactly $r' < r$ outer vertices. Thus none of them are in $S_{=n}^{=r}$. While if G has the rp-fan then the first contractible edge is not (v_1, w_2) and then each $G_o(d(v_1), j)$ is a child triangulation of G if and only if, for each $w = w_2, w_3, \cdots, w_{j-1}$, G has a separating fan containing w and v_1. Such child triangulation is always exist for $j = 3$. □

Now we define a new tree, called *the genealogical tree* T_n^r. We say a vertex v in a tree with root r has *depth* d if the path connecting v and r on the tree has exactly d edges. Define T_n^r as the tree such that (i) the vertices of T_n^r having depth $n-4$ correspond to the triangulations in $S_{=n}^{=r}$, (ii) the vertices of T_n^r having depth at most $n-5$ correspond to the triangulations in $S_n^r \cup S_n^i \cup S_n^o$, and (iii) each edge corresponds to each relation between some G and $P(G)$. The vertex in T_n^r corresponding to K_4 is called *the root* of T_n^r.

Given a triangulation G in $S_n^r \cup S_n^i \cup S_n^o$, we can find all child triangulations of G efficiently, by a similar case analysis to the one in the previous section. If G has k child triangulations, then again we can find them in $O(k)$ time.

Procedure find-all-child-triangulations2(G)
begin
1 Assume G has $n^{'}$ vertices including $r^{'}$ outer vertices.
2 **if** G has exactly n vertices **then** output G **return**
3 **if** $r^{'} < r$ and $r^{'} + (n - n^{'}) > r$
4 **then** { We need more both inner and outer vertices.}
5 **for** each $G_i(i, j)$ with $i \geq 2$ satisfying (a2) or (a3)
6 **find-all-child-triangulations**2($G_i(i, j)$)
7 **for** each $G_o(i, j)$ satisfying (a4)
8 **find-all-child-triangulations**2($G_o(i, j)$)
9 **else if** $r^{'} < r$ and $r^{'} + (n - n^{'}) = r$
10 **then** { We need to increase only outer vertices. }
11 **for** each $G_o(i, j)$ satisfying (a4)
12 **find-all-child-triangulations**2($G_o(i, j)$)
13 **else** $r^{'} = r$ holds
14 **then** { We already have r outer vertices.}
15 { We only need to increase only inner vertices.}
16 **for** each $G_i(i, j)$ satisfying (a1), (a2), (a3) or (b1)
17 **find-all-child-triangulations**2($G_i(i, j)$)
end

Lemma 5. *Let $f(n, r)$ be the number of triangulations in $S^{=r}_{=n}$. Then T^r_n has at most $r \cdot f(n, r)$ vertices.*

Proof. (Sketch) Each vertex of T^r_n corresponds to a triangulation in $S^r_n \cup S^i_n \cup S^o_n \cup S^{=r}_{=n}$. Each vertex with depth $n - 4$ in T^r_n corresponds to a triangulation in $S^{=r}_{=n}$. Each "ancestor" triangulation G of a triangulation $G^{'} \in S^{=r}_{=n}$ has two or more child triangulation if $G \in S^r_n \cup S^i_n$ (since $G_i(1, 2)$ and $G_i(1, 3)$ are always child triangulation of G), and has one or more child triangulation if $G \in S^o_n$ and G has the rp-fan (since $G_i(d(v_1), 3)$ is always child triangulation of G). And each $G \in S^{=r}_{=n}$ has at most $r - 4$ "ancestor triangulations which are in $G \in S^o_n$. Thus T^r_n has at most $r \cdot f(n, r)$ vertices. □

Since the algorithm needs only a constant time of computation for each edge of T^r_n, we have the following lemma.

Lemma 6. *The algorithm uses $O(n)$ space and runs in $O(r \cdot f(n, r))$ time, where $f(n, r)$ is the number of nonisomorphic triconnected rooted plane triangulations having exactly n vertices including exactly r vertices on the outer face.*

We modify our second algorithm further so that it output all triconnected (non-rooted) plane triangulations having exactly n vertices including exactly r vertices on the outer face, as follows.

At each leaf v of the genealogical tree T^r_n, the triangulation G corresponding to v is checked whether the contracting sequence of G with the root vertex is the lexicographically first one among the r contracting sequences of G for r choices of the root vertex on $C_o(G)$, and only if so G is output. Thus we can output only canonical representative of each isomorphism class.

Lemma 7. *The algorithm uses $O(n)$ space and runs in $O(r^2 n \cdot g(n,r))$ time, where $g(n,r)$ is the number of nonisomorphic triconnected (non-rooted) plane triangulations having exactly n vertices including exactly r vertices on the outer face.*

A triconnected (non-rooted) plane triangulations G is *maximal* if G has exactly 3 vertices on $C_o(G)$. For every maximal planar graph by choosing the outer face and the root vertex, there are exactly $2m = 3n - 6$ (triconnected) rooted plane triangulation G with exactly 3 vertices on the outer face, where m is the number of edges of G. We modify the algorithm further as follows. At each leaf v of the genealogical tree T_n^3, the triangulation G corresponding to v is checked whether the contracting sequence of G with the rooted vertex is the lexicographically first one among the $2m$ contracting sequences of G (3 choice of the root vertex on $C_o(G)$ for each of $2m/3$ choice of the outer face of G), and only if so G is output.

Theorem 2. *The modified algorithm generate all maximal planar graphs in $O(n^3 \cdot h(n))$ time, where $h(n)$ is the number of nonisomorphic maximal planar graphs with exactly n vertices. The algorithm uses $O(n)$ space.*

References

[A96] D. Avis, *Generating rooted triangulations without repetitions*, Algorithmica, 16, (1996), pp.618-632.

[B80] T. Beyer and S. M. Hedetniemi, *Constant time generation of rooted trees*, SIAM J. Comput., 9, (1980), pp.706-712.

[G93] L. A. Goldberg, *Efficient algorithms for listing combinatorial structures*, Cambridge University Press, New York, (1993).

[HW74] J. E. Hopcroft and J.K. Wong, *Linear time algorithm for isomorphism of planar graphs*, Proc. of 6th STOC, (1974), pp.172-184.

[KS98] D. L. Kreher and D. R. Stinson, *Combinatorial algorithms*, CRC Press, Boca Raton, (1998).

[LN01] Z. Li and S. Nakano, *Efficient Generation of Plane Triangulations*, ICALP 2001, accepted.

[M98] B. D. McKay, *Isomorph-free exhaustive generation*, J. of Algorithms, 26, (1998), pp.306-324.

[N01] S. Nakano, *Efficient Generation of Triconnected Plane Triangulations*, Technical Report, GU-CS-01-1, Dept. of Comp. Sci., Gunma Univ., http://www.msc.cs.gunma-u.ac.jp/nakano_lab/TR/GU-CS-01-1.ps (2001).

[W86] R. A. Wright, B. Richmond, A. Odlyzko and B. D. McKay, *Constant time generation of free trees*, SIAM J. Comput., 15, (1986), pp.540-548.

Packing Two Disks into a Polygonal Environment*

Prosenjit Bose[1], Pat Morin[2], and Antoine Vigneron[3]

[1] Carleton University. jit@scs.carleton.ca
[2] McGill University. morin@cs.mcgill.ca
[3] Hong Kong University of Science and Technology. antoine@cs.ust.hk

Abstract. We consider the following problem. Given a polygon P, possibly with holes, and having n vertices, compute a pair of equal radius disks that do not intersect each other, are contained in P, and whose radius is maximized. Our main result is a simple randomized algorithm whose expected running time, on worst case input, is $O(n \log n)$. This is optimal in the algebraic decision tree model of computation.

1 Introduction

Let P be a polygon, possibly with holes, and having n vertices. We consider the following problem, which we call 2-DISK: Find a pair of disks with radius r^* that do not intersect each other, are contained in P, and such that r^* is maximized. Biedl *et al.*[5] gave an $O(n^2)$ time algorithm to solve this problem.

Special cases of 2-DISK have been studied previously. When P is a convex polygon, Bose *et al.* [6] describe a linear time algorithm and Kim and Shin [10] describe an $O(n \log n)$ time algorithm. For simple polygons (i.e. polygons without holes), Bespamyatnikh [4] gives an $O(n \log^3 n)$ time algorithm based on the parametric search paradigm [11].

Another special case occurs when the holes of P degenerate to points. This is known as the *maximin 2-site facility location* problem [3,9]. In this formulation we can think of the centers of the two disks as obnoxious facilities such as smokestacks, or nuclear power plants, and the points as population centers. The goal is maximize the distance between each facility and the nearest population center. Katz *et al.* [9] give an $O(n \log n)$ time algorithm for the decision version of the 2-site facility location problem in which one is given a distance d and asked if there exists a placement of 2 non-intersecting disks of radius d, each contained in P such that no point is included in either of the disks.

In this paper we present a simple randomized algorithm for the general case in which P is not necessarily convex and may contain holes. Our algorithm runs in $O(n \log n)$ expected time. It can also be used to solve the optimization version of the 2-site maximin facility location problem in $O(n \log n)$ time. Finally we

* This research was supported by the Natural Sciences and Engineering Research Council of Canada and by the Hong Kong Research Grant Council CERG grant HKUST6137/98E.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 142–149, 2001.
© Springer-Verlag Berlin Heidelberg 2001

observe that, when we allow polygons with holes, $\Omega(n \log n)$ is a lower bound for 2-DISK by a simple reduction from MAX-GAP.

The remainder of the paper is organized as follows: Section 2 reviews definitions and previous results regarding the medial-axis. Section 3 describes our algorithm. Section 4 summarizes and concludes with an open problem.

2 The Medial-Axis

For the remainder of this paper, P will be a polygon, possibly with holes, and having n vertices. The *medial-axis* $M(P)$ of P is the locus of all points p for which there exists a disk centered at p, contained in P, and which intersects the boundary of P in two or more points. See Fig. 1 for an example. Alternatively, $M(P)$ is a portion of the *Voronoi diagram* of the open line segments and vertices defined by the edges of P. To be more precise, we need to remove the Voronoi edges that are outside P and those associated with an edge and one of its endpoints. It is well known that the medial-axis consists of $O(n)$ straight line segments and parabolic arcs.

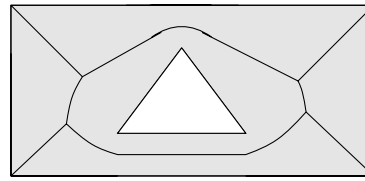


Fig. 1. The medial-axis of a polygon with a triangular hole.

Algorithmically, the medial-axis is well understood. There exists an $O(n)$ time algorithm [7] for computing the medial-axis of a polygon without holes and $O(n \log n)$ time algorithms for computing the medial-axis of a polygon with holes [2]. Furthermore, these algorithms can compute a representation in which each segment or arc is represented as a segment or arc in $\mathbb{R}^3$, where the third dimension gives the radius of the disk that touches two or more points on the boundary of P.

We say that a point $p \in P$ *supports* a disk of radius r if the disk of radius r centered at p is contained in P. We call a vertex, parabolic arc or line segment x of $M(P)$ an *elementary object* if the radius of the largest disk supported by $p \in x$ is monotone as p moves from one endpoint of x to the other. Each edge of $M(P)$ can be split into two elementary objects. Thus, $M(P)$ can be split into $O(n)$ elementary objects whose union is $M(P)$.

3 The Algorithm

Next we describe a randomized algorithm for 2-DISK with $O(n \log n)$ expected running time. We begin by restating 2-DISK as a problem of computing the diameter of a set of elementary objects under a rather unusual distance function. We then use an algorithm based on the work of Clarkson and Shor [8] to solve this problem in the stated time.

The following lemma, of which similar versions appear in Bose *et al.* [6] and Biedl *et al.* [5], tells us that we can restrict our search to disks whose centers lie on $M(P)$.

Lemma 1. *Let D_1 and D_2 be a solution to* 2-DISK *which maximizes the distance between D_1 and D_2 and let p_1 and p_2 be the centers of D_1 and D_2, respectively. Then D_1 and D_2 each intersect the boundary of P in at least two points and hence p_1 and p_2 are points of $M(P)$.*

Proof. Refer to Fig. 2. Suppose that one of the disks, say D_1, intersects the boundary of P in at most one point. Let o_1 be this point, or if D_1 does not intersect the boundary of P at all then let o_1 be any point on the boundary of D_1. Note that there is some value of $\epsilon > 0$ such that D_1 is free to move by a distance of ϵ in either of the two directions perpendicular to the direction $\overrightarrow{p_1 o_1}$ while keeping D_1 in the interior of P. However, movement in at least one of these directions will increase the distance $|p_1 p_2|$, which is a contradiction since this distance was chosen to be maximal over all possible solutions to 2-DISK.

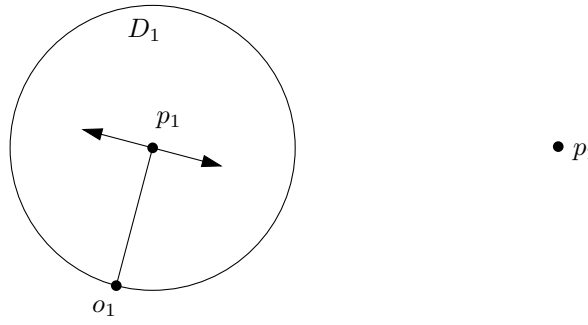


Fig. 2. The proof of Lemma 1

Let x_1 and x_2 be two elementary objects of $M(P)$. We define the *distance* between x_1 and x_2, denoted $d(x_1, x_2)$ as $2r$, where r is the radius of the largest pair of equal-radius non-intersecting disks d_1 and d_2, contained in P and with d_i centered on x_i, for $i = 1, 2$. There are two points to note about this definition of distance: (1) if the distance between two elementary objects is $2r$, then we can place two non-intersecting disks of radius r in P, and (2) the distance from an elementary object to itself is not necessarily 0. Given two elementary objects it

is possible, in constant time, to compute the distance between them as well as the locations of 2 disks that produce this distance [5].

Let E be the set of elementary objects obtained by taking the union of the following three sets of elementary objects:

1. the set of vertices of $M(P)$,
2. the set of elementary line segments obtained by splitting each straight line segment of $M(P)$ into at most two elementary objects.
3. the set of elementary parabolic arcs obtained by splitting each parabolic arc of $M(P)$ into at most two elementary objects.

We call the *diameter* of E the maximum distance between any pair $x, y \in E$, where distance is defined as above. Now, it should be clear from Lemma 1 that 2-DISK can be solved by finding a pair of elements in E whose distance is equal to the diameter of E.[1]

Thus, all that remains is to devise an algorithm for finding the diameter of E. Let m denote the cardinality of E and note that, initially, $m = O(n)$. Motivated by Clarkson and Shor [8], we compute the diameter using the following algorithm. We begin by selecting a random element x from E and finding the element $x' \in E$ whose distance from x is maximal, along with the corresponding radius r. This can be done in $O(m)$ time, since each distance computation between two elementary objects can be done in constant time. Note that r is a lower bound on r^*. We use this lower bound to do *trimming* and *pruning* on the elements of E.

We trim each element $y \in E$ by partitioning y into two subarcs,[2] each of which may be empty. The subarc $y_{\geq}$ is the part of y supporting disks of radius greater than or equal to r. The subarc $y_{<}$ is the remainder of y. We then *trim* $y_{<}$ from y by removing y from E and replacing it with $y_{\geq}$. During the trimming step we also remove from E any element that does not support a disk of radius greater than r. Each such trimming operation can be done in constant time, resulting in an $O(m)$ running time for this step.

Next, we prune E. For any arc $y \in E$, the *lowest point* of y is its closest point to the boundary of P. In the case of ties, we take a point which is closest to one of the endpoints of P. By the definition of elementary objects, the lowest point of y is therefore an endpoint of y. The closed disk with radius r centered on the lowest point of y is denoted by $D(y)$. We discard all the elements $y \in E$ such that $D(y) \cap D(x) \neq \emptyset$ for all $x \in E$.

Pruning can be performed in $O(m \log m)$ time by computing, for each lowest endpoint p, a matching lowest endpoint q whose distance from p is maximal and then discarding p if $|pq| \leq 2r$. This computation is known as *all-pairs furthest neighbors* and can be completed in $O(m \log m)$ time [1].

Once all trimming and pruning is done, we have a new set of elementary objects E' on which we recurse. The recursion completes when $|E'| \leq 2$, at

[1] Here we use the term "pair" loosely, since the diameter may be defined by the distance between an elementary object and itself.

[2] We use the term subarc to mean both parts of segments and parts of parabolic arcs.

which point we compute the diameter of E' in constant time using a brute-force algorithm. We output the largest pair of equal-radius non-overlapping disks found during any iteration of the algorithm.

To prove that this algorithm is correct we consider a pair of non-intersecting disks D_1 and D_2, each contained in P and having radius r^*, centered at p_1 and p_2, respectively, such that the Euclidean distance $|p_1p_2|$ is maximal. The following lemma shows that p_1 and p_2 are not discarded from consideration until an equally good solution is found.

Lemma 2. *If, during the execution of one round, $\{p_1, p_2\} \subset \cup E$ and $r < r^*$, then $\{p_1, p_2\} \subset \cup E'$ at the end of the round.*

Proof. We need to show that at the end of the round, there exists elementary objects $y_1, y_2 \in E'$ such that $p_1 \in y_1$ and $p_2 \in y_2$. More specifically, we need to show there exists $y_1, y_2 \in E$ such that p_1, respectively p_2 is not trimmed from y_1, respectively y_2, and y_1 and y_2 are not pruned.

To see that p_1 and p_2 are not trimmed from any elementary object that contains them we simply note that p_1 and p_2 both support disks of radius $r^* > r$ and are therefore not trimmed.

To prove that y_1 and y_2 are not pruned we subdivide the plane into two open halfspaces H_1 and H_2 such that all points in H_1 are closer to p_1 than to p_2 and vice-versa. We denote by L the line separating these two halfspaces.

Recall that, after trimming, an elementary object x is only pruned if $D(x) \cap D(y) \neq \emptyset$ for all $y \in E$. We will show that $D(y_1) \subseteq H_1$ and $D(y_2) \subseteq H_2$, therefore $D(y_1) \cap D(y_2) = \emptyset$ and neither y_1 nor y_2 are pruned. It suffices to prove that $D(y_1) \subseteq H_1$ since a symmetric argument shows that $D(y_2) \subseteq H_2$. We consider three separate cases depending on the location of p_1 on $M(P)$.

Case 1: p_1 is a vertex of $M(P)$. In this case we choose y_1 to be the singleton elementary object $\{p_1\}$. Thus, $D(y_1)$ is centered at p_1. Furthermore, the distance between p_1 and L is at least $r^* > r$. Therefore, one point of $D(y_1)$ is contained in H_1 and $D(y_1)$ does not intersect the boundary of H_1 so it must be that $D(y_1) \subseteq H_1$.

Case 2: p_1 lies in the interior of a straight line segment of $M(P)$. Let p_1' be the lower endpoint of y_1. Let θ be the angle (p_2p_1, p_1p_1') (see Fig. 3). If $\theta \in [-\pi/2, \pi/2]$ then we can move p_1 slightly in the direction opposite to p_1' while keeping D_1 inside P, thus contradicting the assumption that $|p_1p_2|$ is maximal. Therefore $\theta \in [\pi/2, 3\pi/2]$, which implies that $D(y_1)$ lies in H_1.

Case 3: p_1 lies in the interior of a parabolic arc of $M(P)$. In this case D_1 is tangent to an edge e_1 of P and touches one of its vertices v. p_1' still denotes the lower endpoint of y_1. Without loss of generality, assume that e_1 is parallel to the x–axis and $x(p_1') < x(p_1)$ (see Fig. 4). Let L' be the line parallel to L that crosses the segment $[p_1, p_2]$ and that is tangent to D_1. We denote by o_1 the point where e_1 is tangent to D_1, and we denote by o_1' the point such that (o_1, o_1') is a diameter of D_1. Finally, the convex hull of D_1 and $D(y_1)$ is denoted by $\mathcal{C}$.

It must be that $\mathrm{x}(p_2) > \mathrm{x}(p_1)$, otherwise p_1 and D_1 could be moved in the positive x direction while keeping D_1 in P. This would increase the distance $|p_1p_2|$ which is defined as maximal. It follows that L' is tangent to D_1 along the

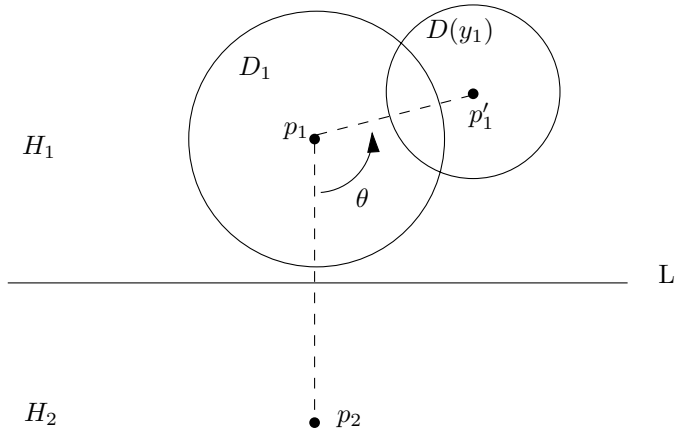


Fig. 3. The proof of Lemma 2 Case 2

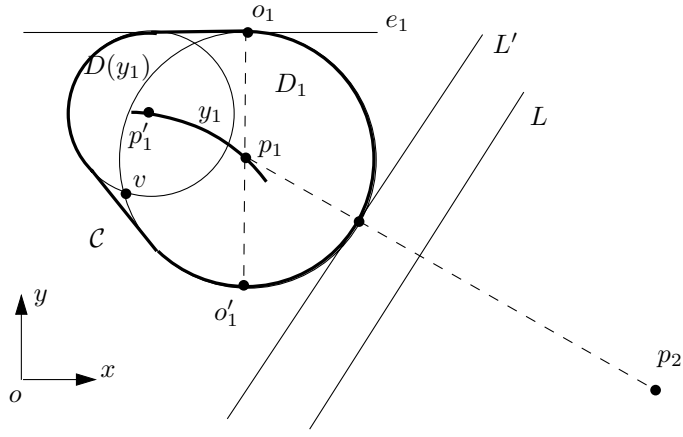


Fig. 4. The proof of Lemma 2 Case 3.

counterclockwise arc (o_1', o_1). Then L' is tangent to $\mathcal{C}$, so by convexity $\mathcal{C}$ lies on the same side of L' as p_1 which implies that D_1 is contained in H_1.

Let d_i denote the distance of the furthest element in E from x_i, and suppose for the sake of analysis that the elements of E are labeled $x_1, \ldots, x_n$ so that $d_i \leq d_{i+1}$. The following lemma helps to establish the running time of the algorithm.

Lemma 3. *If we select $x = x_i$ as the random element, then we discard all $x_j \in E$ such that $j \leq i$ from E.*

Proof. For any $j \leq i$, either x_j does not support a disk of radius greater than d_i, or every point on x_j that supports a disk of radius d_i is of distance at most d_i from any other point of $M(P)$ that supports a disk of radius d_i.

In the first case, x_j is removed from E by trimming. In the second case, $D(x_j) \cap D(x_k) \neq \emptyset$ for all $x_k \in E$ and x_j is removed by pruning.

Finally, we state and prove our main theorem.

Theorem 1. *The above algorithm solves* 2-DISK *in* $O(n \log n)$ *expected time.*

Proof. The algorithm is correct because, by Lemma 2, it never discards p_1 nor p_2 until it has found a solution with $r = r^*$, at which point it has already found an optimal solution that will be reported when the algorithm terminates.

To prove the running time of the algorithm, we use the following facts. Each round of the algorithm can be completed in $O(m \log m)$ time where m is the cardinality of E at the beginning of the round. By Lemma 3, when we select x_i as our random element, all elements x_j with $j \leq i$ disappear from E. Therefore, the expected running time of the algorithm is given by the recurrence

$$T(m) \leq \frac{1}{m} \sum_{i=1}^{m} T(m-i) + O(m \log m) \ ,$$

which readily solves to $O(m \log m)$. Since $m \in O(n)$, this completes the proof.

4 Conclusions

We have given a randomized algorithm for 2-DISK that runs in $O(n \log n)$ expected time. The algorithm is considerably simpler than the $O(n \log^3 n)$ algorithm of Bespamyathnikh [4] and has the additional advantage of solving the more general problem of polygons with holes. Although we have described our algorithm as performing computations with distances, these can be replaced with squared distances to yield an algorithm that uses only algebraic computations.

In the algebraic decision tree model of computation, one can also prove an $\Omega(n \log n)$ lower bound on any algorithm for 2-DISK through a reduction from MAX-GAP [12]. Suppose that the input to MAX-GAP is $y_1, \ldots, y_n$. Without loss of generality one can assume that $y_1 = \min\{y_i : 1 \leq i \leq n\}$ and $y_n = \max\{y_i : 1 \leq i \leq n\}$. We then construct a rectangle with top and bottom sides at y_1 and y_n, respectively, and with width $2(y_n - y_1)$. The interior of this rectangle is then partitioned into rectangles with horizontal line segments having y coordinates $y_1, \ldots, y_n$. See Fig. 5 for an example.

It should then be clear that the solution to 2-DISK for this problem corresponds to placing two disks in the rectangle corresponding to the gap between y_i and y_{i+1} which is maximal, i.e., it gives a solution to the original MAX-GAP problem. Since this reduction can be easily accomplished in linear time and MAX-GAP has an $\Omega(n \log n)$ lower bound, this yields an $\Omega(n \log n)$ lower bound on 2-DISK.

The above reduction only works because we allow polygons with holes. An interesting open problem is that of determining the complexity of 2-DISK when restricted to simple polygons. Is there a linear time algorithm?

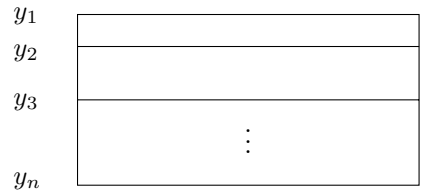


Fig. 5. Reducing MAX-GAP to 2-DISK.

References

1. P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees, and related problems in higher dimensions. *Comput. Geom.: Theory & Appl.*, 4:189–201, 1992.
2. Helmut Alt and Otfried Schwarzkopf. The Voronoi diagram of curved objects. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 89–97, 1995.
3. Boaz Ben-Moshe, Matthew J. Katz, and Michael Segal. Obnoxious facility location: Complete service with minimal harm. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, pages 76–79, 1999.
4. S. Bespamyatnikh. Draft: Efficient algorithm for finding two largest empty circles. In *Proceedings of the 15th European Workshop on Computational Geometry (EuroCG'99)*, pages 37–38, 1999.
5. T. C. Biedl, E. D. Demaine, M. L. Demaine, A. Lubiw, and G. T. Toussaint. Hiding disks in folded polygons. In *Proceedings of the 10th Canadian Conference on Computational Geometry (CCCG'98)*, 1998.
6. P. Bose, J. Czyzowicz, E. Kranakis, and A. Maheshwari. Algorithms for packing two circles in a convex polygon. In *Proceedings of Japan Conference on Discrete and Computational Geometry (JCDCG '98)*, pages 93–103, 1998.
7. F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete and Computational Geometry*, 21, 1999.
8. K. L. Clarkson and P. W. Shor. Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In *Proceedings of the Fourth Annual Symposium on Computational Geometry (SoCG'88)*, pages 12–17, 1988.
9. Matthew J. Katz, Klara Kedem, and Michael Segal. Improved algorithms for placing undesirable facilities. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, pages 65–67, 1999.
10. S. K. Kim and C.-S. Shin. Placing two disks in a convex polygon. *Information Processing Letters*, 73, 2000.
11. N. Megiddo. Applying parallel computation algorithms to the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.
12. F. P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, New York, 1985.

Maximum Red/Blue Interval Matching with Applications*

Danny Z. Chen, Xiaobo (Sharon) Hu, and Xiaodong Wu**

Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{chen, shu, xwu}@cse.nd.edu

Abstract. In this paper, we consider the problem of computing a maximum cardinality matching among a set I of n intervals that are colored as either red or blue, such that a pair of intervals in I can be matched only if they overlap with each other and have different colors. This problem arises in some applications such as radiosurgery treatment planning. We present a greedy algorithm for this problem that runs in $O(n \log \log n)$ time for sorted input. We also solve a useful generalization of this red/blue interval matching problem in the same time bound.

1 Introduction

Consider a set of n input intervals, $I = \{I_1, I_2, \ldots, I_n\}$, on the x-axis that are colored as either red or blue, where each interval $I_i = [le(i), re(i)]$ is specified by its left endpoint $le(i)$ and right endpoint $re(i)$, with $le(i) < re(i)$. Two intervals $I_i = [le(i), re(i)]$ and $I_j = [le(j), re(j)]$ are *disjoint* to each other if $re(i) < le(j)$ or $re(j) < le(i)$; otherwise, they *overlap*. A graph G is called an *interval graph* if there exists a set I_G of intervals such that there is a one-to-one correspondence between the vertices of G and the intervals in I_G and such that any two vertices in G are connected by an edge if and only if their corresponding intervals in I_G overlap. Such an interval set I_G is called an *interval model* of G. An interval graph G is said to be *proper* if and only if there is an interval model I_G of G such that no interval in I_G is contained within any other interval in I_G. Interval graphs find applications in many areas, such as VLSI design, scheduling, biology, medicine, traffic control, and archeology [5]. In this paper, we assume that an interval model of the corresponding interval graph is already given (as this is often the case in applications). We will refer to interval I_i, interval i, interval $[le(i), re(i)]$, and vertex i (corresponding to interval i) interchangeably.

A *matching* in a graph G is a subset M of the edges of G such that no two distinct edges in M are incident to the same vertex. The problem of computing

* The first and the third authors were supported in part by NSF under Grants CCR-9623585 and CCR-9988468. The second author was supported in part by NSF under Grants MIP-9701416 and CCR-9988468.

** Corresponding author.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 150–158, 2001.
© Springer-Verlag Berlin Heidelberg 2001

maximum matchings in graphs has many applications and has received a lot of attention [4].

In this paper, we consider the following matching problem on a set I of n red/blue intervals: Find a maximum cardinality matching M in I such that two intervals can be matched in M only if they overlap and their colors are different. This problem, in fact, is that of computing a maximum cardinality matching in the interval graph G of I that is a bipartite graph [4].

The best known maximum bipartite matching algorithm on a general n-vertex graph is due to Hopcroft and Karp [6], which takes $O(n^{2.5})$ time. To our best knowledge, no algorithm for the specific matching problem on bipartite interval graphs was previously known. There are only a few known results for matching problems on interval graphs and some related graphs. In particular, Moitra and Johnson [7] gave an $O(n \log n)$ time algorithm for maximum matching in ordinary (non-bipartite) interval graphs. For maximum matching in the complement graph of an ordinary interval graph, Andrews and Lee [2] designed an $O(n \log n)$ time algorithm, and Andrews *et al.* [1] provided an $O(n)$ time algorithm for sorted input.

The maximum red/blue interval matching problem that we consider arises in the study of radiosurgery treatment planning. Radiosurgery is a non-invasive technique for eradicating localized benign and malignant tumors by using radiation beams. The shape of a radiation beam is formed by a device called collimator [12]. The distribution of radiation for a treatment is often prescribed by a so-called intensity map of radiation. An intensity map for a multileaf collimator based radiosurgery system [12] may consist of multiple tracks (called arcs). A pair of collimator leaves is assigned to each track. On every track, the prescribed radiation dose is often delivered as a number of (possibly overlapping) sub-tracks, each of which can be viewed as an interval on that track. For example, in Figure 1, we can treat I_j (resp., I_k) as an interval on the track T' (resp., T''). A radiosurgery system usually delivers a unit of radiation to a portion of a track as specified by one of its sub-tracks, with its two collimator leaves defining the two endpoints of such a sub-track. For two consecutive tracks, such as T' and T'' in Figure 1, suppose that there are two sub-tracks, one on each track, whose intervals overlap (here, although the intervals are really on different tracks, we consider their overlapping as if the intervals were all on the x-axis). Then the two corresponding sub-tracks on the two consecutive tracks can be delivered by the radiosurgery system in the same time, thus shortening the total treatment time. (The overlapping among the sub-tracks on the same track does not help since they must be delivered one at a time.) Suppose we color the intervals on T' (resp., T'') as red (resp., blue). Then one of the useful problems in treatment planning for intensity-modulated radiation therapy with multileaf collimation [3,12] is to obtain the maximum number of red/blue interval pairs such that the two intervals in each pair overlap. This gives rise to our maximum red/blue interval matching problem.

We also need to study a slightly more general version of the maximum red/blue interval matching problem. The endpoints of the sub-tracks (intervals)

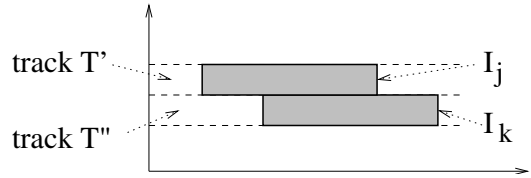


Fig. 1. Illustrating an application of the red/blue interval matching problem.

in the above radiosurgery treatment planning setting are specified by pairs of leaves of a collimator. Due to some mechanical constraints (e.g., for preventing two opposite collimator leaves on two consecutive tracks from colliding into each other and being damaged), a system-dependent threshold value $d > 0$ is actually specified for the overlapping of any two red/blue intervals [11]. Precisely, the length of the overlapping portion of any two such intervals must be no smaller than the threshold value d. Hence, the *threshold overlapping maximum red/blue interval matching problem* is to find a maximum matching of n red/blue intervals subject to that two red/blue intervals can be matched only if the length of their overlapping portion is $\geq d$. Obviously, this problem is a generalized version of the maximum red/blue interval matching problem.

We present an $O(n \log\log n)$ time algorithm for the maximum red/blue interval matching problem if the endpoints of the input intervals are already given sorted; otherwise, the algorithm takes $O(n \log n)$ time. Our algorithm is of a greedy nature, and is based on the plane sweeping technique [8] and some integer data structures [9,10]. For the special case of a proper interval graph with sorted interval endpoints, we give an $O(n)$ time matching algorithm. We also show an easy $O(n)$ time reduction from the threshold overlapping maximum red/blue interval matching problem to the maximum red/blue interval matching problem for sorted input.

The rest of this paper is organized as follows. Section 2 gives some needed preliminaries. Section 3 presents our maximum red/blue interval matching algorithms. Section 4 shows the reduction from the threshold overlapping maximum red/blue interval matching problem to the maximum red/blue interval matching problem.

2 Preliminaries

The input consists of a set of n red or blue intervals, $I = \{I_1, I_2, \ldots, I_n\}$. The endpoints of the intervals in I are arbitrary numbers on the x-axis. Each I_i is associated with a color $C(I_i) \in \{red, blue\}$. To avoid cluttering the exposition, we assume without loss of generality (WLOG) that no two input intervals share the same endpoint (i.e., the $2n$ endpoints are distinct). Our algorithms can be easily modified for the general case.

We first sort the $2n$ endpoints of I from left to right if they are not given sorted. The sorting takes $O(n \log n)$ time. From now on, unless otherwise spec-

ified, we assume that the $2n$ endpoints of I are available in this sorted order and are stored in an array A. Further, we treat each interval endpoint of I as its *rank* in this sorted array A, i.e., we simply treat each interval endpoint as an integer in $\{1, 2, \ldots, 2n\}$. We also assume WLOG that the intervals in I have been relabeled such that $i < j$ implies that $re(i)$ occurs before $re(j)$ in the sorted array A of endpoints.

If the input interval graph is proper, then we assume that no interval in I contains another interval in I.

Given a matching M in I, we say that an interval i is *in* M if i is matched by M. An interval is *free* if it is unmatched with respect to M. Interval i matching with j in M is denoted as $mate_M(i) = j$. A matched pair of intervals i and j in M is denoted as $match_M(i, j)$.

It is known that when the keys of a priority queue are all from a universe $\{1, 2, \ldots, m\}$, there is a data structure which supports each of the following operations in $O(\log \log m)$ time [9,10]: **Insert**, **Delete**, **Member**, **Empty**, **Min**, **Max**, **Predecessor**, and **Successor**. Our algorithm makes use of such integer priority queues (with $m = 2n$).

3 The Maximum Red/Blue Matching Algorithms

Our maximum red/blue interval matching algorithm is based on the following key observation.

Lemma 1. *Let J be the interval in I such that its right endpoint $re(J)$ is the smallest among the right endpoints of all intervals in I. WLOG, let $C(J) = red$ (the case of $C(J) = blue$ is symmetric). Then the following holds.*

(i) If J does not overlap with any blue interval in I, then J is free in any matching M in I.

(ii) If J overlaps with any blue interval J' in I, then $re(J) < re(J')$. Further, let J'' be the blue interval in I that overlaps with J and whose right endpoint $re(J'')$ is the smallest among all blue intervals of I that overlap with J. Then for any maximum matching M^* in I, there is another maximum matching M' in I such that $mate_{M'}(J) = J''$.

Proof: Since the other statements of this lemma are easy to show, we only prove that for any maximum matching M^* in I, there is another maximum matching M' in I such that $mate_{M'}(J) = J''$. Let M^* be any maximum matching in I. Note that it is impossible that both J and J'' are free with respect to M^* (otherwise, we can add (J, J'') to M^*, a contradiction to the optimality of M^*). WLOG, we assume that the pair $(J, J'') \notin M^*$ (otherwise, $M' = M^*$). We consider three cases: (1) J is free and J'' is in M^*, (2) J is in M^* and J'' is free, and (3) both J and J'' are in M^* (but $(J, J'') \notin M^*$). We will show that in each case, there is another maximum matching M' in I such that $mate_{M'}(J) = J''$.

It is easy to obtain a maximum matching M' in I from M^* for Cases (1) and (2), as follows: Replace $match_{M^*}(J', J'')$ by $match_{M'}(J, J'')$ in Case (1), and replace $match_{M^*}(J, J')$ by $match_{M'}(J, J'')$ in Case (2), where $J' \notin \{J, J''\}$

is the interval (with an appropriate color) that is matched with J'' or J in M^* in these two cases, respectively; let M' keep the other matched pairs of M^* unchanged.

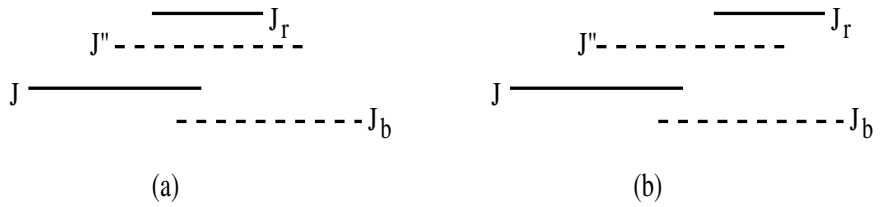


Fig. 2. Obtaining M' from M^* by replacing $match_{M^*}(J, J_b)$ and $match_{M^*}(J'', J_r)$ by $match_{M'}(J, J'')$ and $match_{M'}(J_r, J_b)$ (the solid line segments are red intervals and the dashed line segments are blue intervals).

For Case (3), let $mate_{M^*}(J) = J_b$ and $mate_{M^*}(J'') = J_r$, where $C(J_b) = blue$ and $C(J_r) = red$. We aim to prove that J_b and J_r overlap (and hence can be matched). By the definitions of J and J'', we have $re(J) < re(J'') < re(J_b)$ (see Figure 2). This and the fact that J and J_b overlap imply that the interval $[re(J), re(J'')]$ on the x-axis is completely contained by J_b. Also, by the definition of J, we have $re(J) < re(J_r)$. Hence, either $re(J) < re(J_r) < re(J'')$ (see Figure 2(a)) or $re(J) < re(J'') < re(J_r)$ (see Figure 2(b)) holds. In either situation, due to the fact that J'' and J_r overlap, J_r must overlap with the interval $[re(J), re(J'')]$. Hence, J_r and J_b must overlap (precisely, on at least one endpoint of the interval $[re(J), re(J'')]$), and can be matched. Therefore, we can obtain a maximum matching M' in I from M^* by replacing $match_{M^*}(J, J_b)$ and $match_{M^*}(J'', J_r)$ by $match_{M'}(J, J'')$ and $match_{M'}(J_r, J_b)$, and letting M' keep the other matched pairs of M^* unchanged. This finishes the proof of this lemma. □

Lemma 1 immediately implies the following greedy algorithm: Find intervals J and J'' in I as defined in Lemma 1, let them form a matched pair, and solve the problem recursively on the remaining intervals in $I - \{J, J''\}$. Of course, an efficient way to carry out this greedy method is by using a non-recursive algorithm. Indeed, our greedy algorithm is based on the plane sweeping technique [8] and integer priority queues [9,10].

We first give an overview of our algorithm. It uses a sweep line to scan the endpoints of the intervals of I in the increasing order. It maintains two integer priority queues, Q_R (for the red intervals) and Q_B (for the blue intervals), such that all intervals in Q_R or Q_B are currently being intersected by the sweep line. Note that if at the same time, a red (resp., blue) interval J' (resp., J'') is stored in Q_R (resp., Q_B), then J' and J'' overlap and are possible to form a matched pair. An interval J is inserted into (resp., deleted from) the corresponding priority queue when its left (resp., right) endpoint $le(J)$ (resp., $re(J)$) is encountered by the sweep line.

The details of our algorithm follow next.

Algorithm Max-RB-Matching(I);

1. Initialize the integer priority queues Q_R and Q_B as containing no key values. Let L be the sorted list of all endpoints of I in the increasing order. Let the output maximum matching in I initially be $M^* = \phi$. Mark every interval of I as "free".
2. While L is not empty do
   (a) Remove an endpoint p from the head of L. Let J be the interval of I such that p is an endpoint of J. WLOG, assume that $C(J) = red$ (the other case is symmetric).
   (b) If p is the left endpoint $le(J)$ of J, then insert the right endpoint $re(J)$ of J into the priority queue Q_R. (Note that $re(J)$ is not removed from the list L.) Go to the next iteration.
   (c) If J is already in M^*, then go to the next iteration.
   Else, p is the right endpoint $re(J)$ of J. Search in Q_B for the immediate successor $Succ_{Q_B}(re(J))$ of $re(J)$ in Q_B.
   If $Succ_{Q_B}(re(J))$ does not exist in Q_B, then Q_B contains no blue interval. Delete $re(J)$ from Q_R. (The interval J is not matched.)
   If $Succ_{Q_B}(re(J))$ does exist in Q_B, then let it be the right endpoint $re(J'')$ of a blue interval J''. Add a matched pair (J, J'') to the output maximum matching M^*. Delete $re(J)$ from Q_R and $re(J'')$ from Q_B. Mark both J and J'' as "matched".
   End of the while loop.

Lemma 2. Algorithm Max-RB-Matching *computes a maximum matching for n red and blue intervals in $O(n \log\log n)$ time, provided that the endpoints of the intervals are already given sorted.*

Proof: The correctness of the algorithm follows from Lemma 1. WLOG, we only consider the situation when the right endpoint $re(J)$ of a red interval J is encountered by the sweep line, and further, we assume that J is currently not in M^*. Note that all blue intervals of I that are currently being intersected by the sweep line and that are not in M^* are stored in the priority queue Q_B. If no blue interval is in Q_B (as found by the search of $Succ_{Q_B}(re(J))$ in Q_B), then by Lemma 1, J should be free. If the immediate successor $Succ_{Q_B}(re(J))$ of $re(J)$ in Q_B is obtained, then again by Lemma 1, (J, J'') should be matched in M^*, where J'' is the blue interval having $Succ_{Q_B}(re(J))$ as its right endpoint.

Analyzing the time complexity of **Algorithm Max-RB-Matching** is easy. Each of the n right endpoints of the n input intervals is inserted once into an integer priority queue and deleted once from such a queue, and it gives rise to at most one **Successor** search in such a queue. Since each of the **Insert**, **Delete**, and **Successor** operations takes $O(\log\log n)$ time [9,10], the total running time of the algorithm is $O(n \log\log n)$. □

We now consider the proper interval graph case (no interval in I contains another interval in I).

Lemma 3. *A maximum matching for a set I of n red and blue intervals can be computed in $O(n)$ time, provided that the endpoints of the intervals in I are already given sorted and no interval in I contains another interval in I.*

Proof: This follows quite easily from Lemma 1. We maintain two sorted lists I_R and I_B, with I_R (resp., I_B) containing the red (resp., blue) intervals in I ($I_R \cup I_B = I$). Note that since no interval in I contains another interval in I, we can order the intervals in I_R and I_B based only on their left endpoints.

Our $O(n)$ time algorithm for computing a maximum matching in I is very similar to merging two sorted lists. We consider the first element J of I_R and the first element J' of I_B. If J and J' overlap, then by Lemma 1, we match J and J' and remove J from I_R and remove J' from I_B. Otherwise, J and J' do not overlap with each other. Assume WLOG that $le(J) < le(J')$. Then by Lemma 1 again, we remove J from I_R (but J' stays in I_B). In either case, we repeat the matching process on the remaining elements of I_R and I_B, until one of I_R or I_B becomes empty. It is easy to see that since I_R and I_B are both sorted, this algorithm takes $O(n)$ time. □

4 Threshold Overlapping Maximum Red/Blue Matching

As a generalization of the maximum red/blue interval matching problem, we solve the threshold overlapping maximum red/blue interval matching problem by reducing it to the maximum red/blue interval matching problem. Again, we assume that the endpoints of I are already given sorted. However, at this section, we need to use the original input values of the endpoints, not their integral ranks as in Section 3.

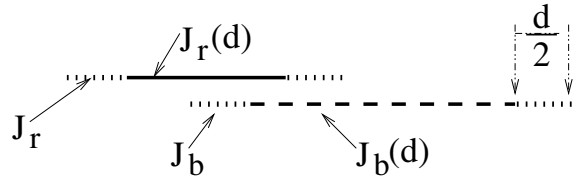


Fig. 3. Illustrating Lemma 4 (the dotted line segments are the cut-away portions).

Let $d > 0$ be the specified threshold value for the minimum acceptable length of overlapping between any two red/blue intervals. The following simple observation is a key to our reduction.

Lemma 4. *For any red (resp., blue) interval J_r (resp., J_b), let $J_r(d)$ (resp., $J_b(d)$) be the interval obtained from J_r (resp., J_b) by cutting a length of $d/2$ away from each end of J_r (resp., J_b). Then J_r and J_b have an overlapping length $\geq d$ if and only if $J_r(d)$ and $J_b(d)$ overlap (see Figure 3).*

Proof: Easy and omitted. □

Note that if an interval J of I has a length $< d$, then applying to J the cut-away operation as defined in Lemma 4 creates an *empty interval* $J(d)$ that we should throw away from further consideration (since J will never overlap with any interval in a length $\geq d$). We say that such an interval J is *short*. In fact, all short intervals of I are not good and should be identified, because the distances between their endpoints are less than the threshold value d. (Thus, they possibly cause the opposite collimator leaves on the same tracks, as well as on consecutive tracks, that define the endpoints of these and other intervals to collide into each other [11].) Hence, all short intervals of I are not useful medically and should be removed even before one considers solving the threshold overlapping maximum red/blue interval matching problem. WLOG, we assume that there are no short intervals in I. Let $I(d)$ be the set of intervals obtained from I by applying the cut-away operation of Lemma 4.

All that is left for us to do is to put the endpoints of $I(d)$ into a sorted order. Note that the cut-away operations may cause the order of the endpoints of I to be different from the order of the endpoints of $I(d)$, and we want to sort the endpoints of $I(d)$ in $O(n)$ time.

Let $LE(I)$ (resp., $LE(I(d))$) be the list of left endpoints of I (resp., $I(d)$), and $RE(I)$ (resp., $RE(I(d))$) be the list of right endpoints of I (resp., $I(d)$). Note that both $LE(I)$ and $RE(I)$ are already in a sorted order. Since the cut-away operations basically shift all points in $LE(I)$ (resp., $RE(I)$) to the right (resp., left) by the same distance of $d/2$, $LE(I(d))$ and $RE(I(d))$ are in the same sorted order as $LE(I)$ and $RE(I)$, respectively. Hence, to create a sorted list for all endpoints of $I(d)$, we simply merge $LE(I(d))$ and $RE(I(d))$, in $O(n)$ time. This finishes our reduction from the threshold overlapping maximum red/blue interval matching problem to the maximum red/blue interval matching problem.

Lemma 5. *In $O(n)$ time, the threshold overlapping maximum red/blue interval matching problem can be reduced to the maximum red/blue interval matching problem, if the endpoints of the n input intervals are already given in a sorted order.*

Acknowledgement

The authors would like to thank Dr. Cedric Yu, Department of Radiation Oncology, School of Medicine, University of Maryland at Baltimore, USA, for discussing the medical motivations of this and other related radiosurgery treatment planning problems with us.

References

1. M. G. Andrews, M. J. Atallah, D. Z. Chen, and D. T. Lee, "Parallel algorithms for maximum matching in complements of interval graphs and related problems," *Algorithmica*, 26 (2) (2000), pp. 263–289.

2. M. G. Andrews and D. T. Lee, "An optimal algorithm for matching in interval graphs," manuscript, 1992.
3. D. Z. Chen, X. Wu, L. Zhang, X. Hu, and C. Yu, "A new algorithm for intensity-modulated arc therapy (IMAT) with dynamic multileaf collimation," manuscript, 2001.
4. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, New York, NY, 1990.
5. M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
6. J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM J. Comput.*, 2 (1973), pp. 225–231.
7. A. Moitra and R. C. Johnson, "A parallel algorithm for maximum matching in interval graphs," *Proc. Int. Conf. Parallel Processing*, 1989, Vol. III, pp. 114–120.
8. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
9. P. van Emde Boas, "Preserving order in a forest in less than logarithmic time and linear space," *Information Processing Letters*, 6 (3) (1977), pp. 80–82.
10. P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical Systems Theory*, 10 (1977), pp. 99–127.
11. C. X. Yu, personal communication, January 2001.
12. C. X. Yu, D. Yan, M. N. Du, S. Zhou, and L. J. Verhey, "Optimization of leaf positioning when shaping a radiation field with a multileaf collimator," *Phys. Med. Biol.*, 40 (2) (1995), pp. 305–308.

Computing Farthest Neighbors on a Convex Polytope*

Otfried Cheong[1], Chan-Su Shin[2], and Antoine Vigneron[3]

[1] Institute of Information and Computing Sciences,
Utrecht University, Netherlands.
otfried@cs.uu.nl
[2] Department of Computer Science,
KAIST, Korea.
cssin@jupiter.kaist.ac.kr
[3] Department of Computer Science,
The Hong Kong University of Science and Technology, China.
antoine@cs.ust.hk

Abstract. Let N be a set of n points in convex position in $\mathbb{R}^3$. The farthest-point Voronoi diagram of N partitions $\mathbb{R}^3$ into n convex cells. We consider the intersection $G(N)$ of the diagram with the boundary of the convex hull of N. We give an algorithm that computes an implicit representation of $G(N)$ in expected $O(n \log^2 n)$ time. More precisely, we compute the combinatorial structure of $G(N)$, the coordinates of its vertices, and the equation of the plane defining each edge of $G(N)$. The algorithm allows us to solve the all-pairs farthest neighbor problem for N in expected time $O(n \log^2 n)$, and to perform farthest-neighbor queries on N in $O(\log^2 n)$ time with high probability. This can be applied to find a Euclidean maximum spanning tree and a diameter 2-clustering of N in expected $O(n \log^4 n)$ time.

1 Introduction

Let N be a set of n points in three dimensions. Its diameter is the maximum distance between any two points in N. The problem of computing the diameter has been intensively studied in the past two decades. Indeed, back in 1985 Preparata and Shamos [6] called it a source of frustration to many workers. After Clarkson and Shor [4] gave a simple randomized algorithm that runs in optimal expected $O(n \log n)$ time, most work on the problem has concentrated on finding a matching deterministic algorithm. After considerable efforts by several researchers, Ramos [7] and Bespamyatnikh [3] achieved deterministic algorithms that run in $O(n \log^2 n)$ time. Finally, Ramos [8] solved the problem in optimal time $O(n \log n)$.

* This research was partially supported by the Hong Kong Research Grants Council. Part of it was done when the first two authors were at HKUST. The second author's research was supported by grant No. 98-0102-07-01-3 from KOSEF and also supported by BK21 Research Professor Program at KAIST.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 159–169, 2001.
© Springer-Verlag Berlin Heidelberg 2001

The all-pairs farthest neighbors problem for a set N of n points in three dimensions is to compute, for each point p in N, the point of N farthest from p. This natural generalization of the diameter problem has several applications [1]. While all-pairs *nearest* neighbors in fixed dimension d can be computed in optimal $O(n \log n)$ time [9], no algorithm with similar efficiency is known for the all-pairs *farthest* neighbors. Agarwal et al. [1] showed that 3-dimensional all-pairs farthest neighbors can be computed in $O(n^{4/3} \log^{4/3} n)$ time, and pose closing the gap between this and the only lower bound of $\Omega(n \log n)$ as a challenging open problem.

Our work is motivated by the fact that two-dimensional all-pairs farthest neighbors can be computed in linear time if the points are the vertices of a given convex polygon, even though the problem has complexity $\Omega(n \log n)$ for arbitrary points [2]. We consider the all-pairs farthest neighbor problem for a set of n points that form the vertices of a convex polytope in 3 dimensions, and show that it can be solved in expected $O(n \log^2 n)$ time. This is the first algorithm on a restricted all-pairs farthest neighbor problem in three dimensions that runs in near-linear time.

Our algorithm computes a data structure that is interesting in its own right: given a convex polytope $\mathcal{P}$ with m vertices, and n sites lying on its surface, we compute an implicit representation of the intersection of the farthest-point Voronoi diagram of these sites with the surface of $\mathcal{P}$ in expected $O(m \log m + n \log n \log m)$ time.

Our result can be used to solve the following problems, as a direct application of Agarwal et al.'s results [1]:

- *Bi-chromatic farthest neighbors*: Given a set R of n "red" points and another set B of m "blue" points in 3 dimensions such that $R \cup B$ is in convex position, we find for each red point $r \in R$ the farthest blue point from r in expected $O((n+m) \log^2(n+m))$ time.
- *External farthest neighbors*: Given a set N of n points in 3 dimension in convex position and its partition $N_1, N_2, \ldots, N_m$ into m subsets, we compute in expected $O(n \log^3 n)$ time for each point p in N, a farthest point in $N \backslash N_i$, where $p \in N_i$.
- *Euclidean maximum spanning tree*: Given a set N of n points in 3 dimensions in convex position, we compute in expected $O(n \log^4 n)$ time a spanning tree of N whose edges have the maximum total length among all spanning trees, where the length of an edge is the Euclidean distance between its endpoints. From this tree we can compute a minimum diameter 2-clustering of N in linear time.

In the remainder of the paper, we consider only the original problem of computing all-pairs farthest neighbors for a set of points in convex position.

2 Preliminaries

Given a set N of points *sites* in $\mathbb{R}^3$ and a point site s not necessarily in N, we define the (farthest-point) *Voronoi cell* $\mathrm{Vor}(s|N)$ of s with respect to N as the

set of points $x \in \mathbb{R}^3$ such that the Euclidean distance $d(x, s)$ is larger than the distance $d(x, s')$ to any site $s' \in N$ with $s' \neq s$. Voronoi cells are convex, and may be empty. The (farthest-point) *Voronoi diagram* of N is the partition of $\mathbb{R}^3$ into the Voronoi cells $\mathrm{Vor}(s|N)$, for $s \in N$.

Let now $\mathcal{P}$ be the *boundary* of a convex polytope in three dimensions. Let N be a set of point sites lying on $\mathcal{P}$, and s a site on $\mathcal{P}$ not necessarily in N. The Voronoi cell $\mathrm{Vor}(s|N)$ intersects $\mathcal{P}$ in a two-dimensional, possibly empty, *Voronoi face* $\mathrm{Vor}_{\mathcal{P}}(s|N)$. Our first observation is that Voronoi faces are simply connected.

Lemma 1. *Let $\mathcal{P}$ be the boundary of a 3-dimensional polytope, N a set of point sites on $\mathcal{P}$, and $s \in N$. The Voronoi face $\mathrm{Vor}_{\mathcal{P}}(s|N)$ is simply connected, that is, its boundary is a simple closed curve.*

Proof. Let p, q be two points in $\mathrm{Vor}_{\mathcal{P}}(s|N)$. Let $C_s(pq)$ be the two-dimensional cone with apex s spanned by pq, and let $L_s(pq)$ be the intersection $C_s(pq) \cap \mathcal{P}$. $L_s(pq)$ is a path on $\mathcal{P}$ connecting p and q. We prove that $L_s(pq)$ lies entirely in $\mathrm{Vor}_{\mathcal{P}}(s|N)$.

In fact, let $x \in pq$. Since $x \in \mathrm{Vor}(s|N)$, N lies in the sphere S with center x and passing through s. If we enlarge S by moving its center along the ray sx and keeping s on the sphere, N will remain inside the enlarged sphere. It follows that the entire portion of $C_s(pq)$ not in the triangle spq is contained in $\mathrm{Vor}(s|N)$. Since $L_s(pq)$ lies in this portion, we have $L_s(pq) \subset \mathrm{Vor}_{\mathcal{P}}(s|N)$.

Moreover, if we centrally project $L_s(pq)$ from s, the result is a line segment. Since for any two points p, q in $\mathrm{Vor}_{\mathcal{P}}(s|N)$ we have $L_s(pq) \subset \mathrm{Vor}_{\mathcal{P}}(s|N)$, this implies that the central projection of $\mathrm{Vor}_{\mathcal{P}}(s|N)$ from s is convex, and therefore simply connected.

It follows that a set of sites N on $\mathcal{P}$ partitions $\mathcal{P}$ into simply connected faces, defining a planar graph that we denote as $G(N) = G_{\mathcal{P}}(N)$. A face of $G(N)$ is a Voronoi face, a vertex of $G(N)$ is a point of equal distance from three sites, and therefore the intersection of an edge of the three-dimensional Voronoi diagram with $\mathcal{P}$. There can be at most two vertices defined by the same three sites (and this case can indeed arise). An edge of $G(N)$ separates two Voronoi faces $\mathrm{Vor}_{\mathcal{P}}(s|N)$ and $\mathrm{Vor}_{\mathcal{P}}(s'|N)$, and therefore lies on the bisecting plane of the sites s and s'.

Theorem 1. *Let N be a set of n sites on a polytope $\mathcal{P}$. Then $G(N) = G_{\mathcal{P}}(N)$ has $O(n)$ vertices, edges, and faces.*

Proof. It follows from Lemma 1 that the number of faces is at most n. A vertex has degree at least three. Euler's formula now implies the linear bound on the number of vertices and edges.

The embedding of an edge e of $G(N)$ in $\mathcal{P}$ is a polyline whose vertices are the intersections between the embedding of e and the edges of $\mathcal{P}$. If $\mathcal{P}$ has m edges, the embedding of e consists of at most m segments. The overall complexity of

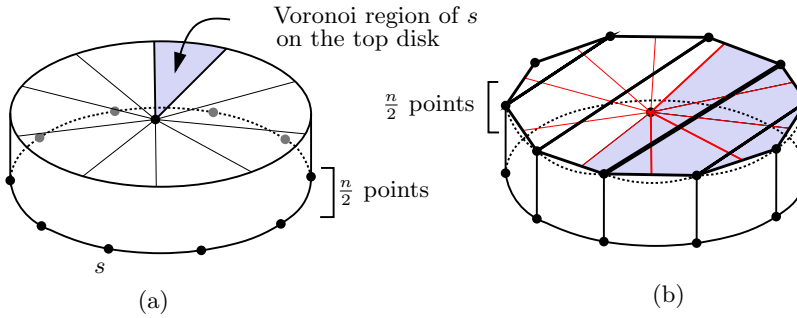


Fig. 1. An example of $G(N)$ with complexity of $\Omega(n^2)$. $\mathcal{P}$ is the convex hull of N. (a) If one puts $n/2$ points equidistantly on the boundary of the bottom disk, then their Voronoi cells partition the top disk equally. (b) Add a convex roof slightly above the top disk which consists of the remaining $n/2$ points. The number of intersections of the Voronoi cells of the points on the bottom disk with edges of the roof becomes $\Omega(n^2)$. The fat edge of the roof intersects the shaded Voronoi cells.

the embedding of $G(N)$ is therefore $O(nm)$. This bound is tight, as the example of the modified n-Camembert in Fig. 1 shows.

To achieve subquadratic time, we cannot work with the explicit embedding of $G(N)$ into $\mathcal{P}$. Instead, we will use a linear size representation of $G(N)$. The representation stores the adjacency relations between vertices, edges, and faces of $G(N)$. By Theorem 1 this has linear complexity. In addition, we record for each face the defining site, and for each vertex the coordinates of its embedding into $\mathcal{P}$.

We need the following lemma.

Lemma 2. *Let N be a set of sites on a polytope $\mathcal{P}$, and let v be a vertex of $G(N) = G_{\mathcal{P}}(N)$. If the sites defining the faces around v are known in the correct order, then the coordinates of v's embedding into $\mathcal{P}$ can be computed using a single ray shooting query on $\mathcal{P}$.*

Proof. The sites define a line of equal distance, which intersects $\mathcal{P}$ in two points. The two points differ in the order of sites. The order of sites thus orients the line, and we can find the correct intersection point with $\mathcal{P}$ using a ray shooting query from infinity.

As a final observation, note that this approach to computing the farthest–point
Voronoi diagram would fail if we tried to apply it to the *nearest-point* Voronoi diagram instead. In fact, the intersection of the nearest-point Voronoi diagram of n sites N with the boundary of the convex hull of N can have a quadratic number of faces; such an example can be easily designed.

In the following we will assume N to be in *general position*, which means that no five sites lie on a sphere and no four points lie on a circle.

3 The Strategy to Compute $G(N)$

Given a convex polytope $\mathcal{P}$ with m vertices and a set N of n point sites on the surface $\mathcal{P}$, we show how to compute $G(N) = G_{\mathcal{P}}(N)$ in expected time $O(m \log m + n \log n \log m)$.

The first step of the algorithm is to compute, in $O(m \log m)$ time, a data structure that permits ray shooting queries in $\mathcal{P}$ with query time $O(\log m)$.

We then compute $G(N)$ by randomized incremental construction. We choose a random permutation $s_1, s_2, s_3, \ldots, s_n$ of N, and insert the sites in this order.

Let $N^i = \{s_1, s_2, ...s_i\}$ be the set of the first i inserted sites. The algorithm maintains the implicit representation of $G(N^i)$ while sites are added one by one, resulting in $G(N^n) = G(N)$. Note that the polytope $\mathcal{P}$ defining $G(N^i)$ does not change during the course of the algorithm.

Our algorithm is mostly a straightforward implementation of the randomized incremental paradigm using a conflict graph, and most of the lemmas below are analogous to those proven, say, in Mulmuley's book [5]. We do, however, need to cope with an unpleasant aspect of our diagram concerning the maintenance of the conflict graph. As we will see, conflicts can "jump" to another edge. We need $O(\log m)$ time to check a conflict between a site and an edge, which results in overall $O(n \log n \log m)$ expected time for the computation of $G(N)$.

4 Conflicts

A vertex v of $G(N^i)$ is said to be in *conflict* with a site $s \in N \setminus N^i$ if s is farther from v than any of the sites that define v, that is, the sites whose faces are adjacent to v. This is equivalent to $v \in \mathrm{Vor}_{\mathcal{P}}(s|N^i)$. Similarly, an edge e of $G(N^i)$ is said to be in *conflict* with a site $s \in N \setminus N^i$ if $e \cap \mathrm{Vor}_{\mathcal{P}}(s|N^i) \neq \emptyset$.

In addition to a representation of $G(N^i)$, our algorithm maintains a conflict list: for each not-yet-inserted site $s \in N \setminus N^i$, we keep a bidirectional pointer to a single vertex $X(s)$ of $G(N^i)$ in conflict with s. If no vertex of $G(N^i)$ is in conflict with s, we set $X(s)$ to a single edge of $G(N^i)$ in conflict with s. If no edge of $G(N^i)$ is in conflict with s either, then $X(s) := \emptyset$.

Lemma 3. *Let $s \in N \setminus N^i$. If $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ is not empty, then the vertices and edges of $G(N^i)$ in conflict with s form a connected subgraph of $G(N^i)$.*

Proof. Suppose the vertices and edges are not connected. Then we can separate them using a curve γ contained in $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ that cuts $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ into two non-empty components without intersecting vertices or edges of $G(N^i)$. This means that γ is entirely contained in a face $\mathrm{Vor}_{\mathcal{P}}(s'|N^i)$ of $G(N^i)$. Since $\mathrm{Vor}_{\mathcal{P}}(s'|N^i)$ is simply connected by Lemma 1, γ cannot be a closed curve. The endpoints of γ lie in $\mathrm{Vor}_{\mathcal{P}}(s'|N^i) \setminus \mathrm{Vor}_{\mathcal{P}}(s|N^i) = \mathrm{Vor}_{\mathcal{P}}(s'|N^i \cup \{s\})$. Since this set is connected by Lemma 1, there is a path γ' connecting the endpoints of γ through $\mathrm{Vor}_{\mathcal{P}}(s'|N^i) \setminus \mathrm{Vor}_{\mathcal{P}}(s|N^i)$. The concatenation of γ and γ' is a closed curve contained in $\mathrm{Vor}_{\mathcal{P}}(s'|N^i)$. It separates the edges and vertices of $G(N^i)$, a contradiction to Lemma 1.

We consider now the insertion of the $i+1$'st site s_{i+1} into the data structure storing $G(N^i)$.

Lemma 4. *During the insertion of s_{i+1}, one of the three following situations occurs (see Fig. 3):*

(i) *If s_{i+1} has no conflicting vertex and no conflicting edge in $G(N^i)$, then $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) = \emptyset$ and $G(N^{i+1}) = G(N^i)$.*
(ii) *If s_{i+1} has only one conflicting edge e and no conflicting vertex in $G(N^i)$, then $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \cap G(N^i)$ is a connected portion pq of e.*
(iii) *If s_{i+1} has at least one conflicting vertex in $G(N^i)$, then at least one endpoint of each edge in conflict with s_{i+1} is also in conflict with s_{i+1}.*

Proof. If s_{i+1} has no conflicting vertex and no conflicting edge in $G(N^i)$, then $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ must be empty; otherwise, $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ would lie entirely within some face of $G(N^i)$, which is impossible since the face is simply connected in $G(N^{i+1})$. Thus if $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \neq \emptyset$, then there must be a vertex or an edge of $G(N^i)$ in conflict with s_{i+1}.

Suppose now that an edge of $G(N^i)$ conflicts with s_{i+1}, but that none of its endpoints conflicts with s_{i+1}. Then Lemma 3 directly implies that no other vertex or edge of $G(N^i)$ is in conflict with s_{i+1}.

The third case trivially holds because the vertices and edges in conflict with s_{i+1} form a connected subgraph of $G(N^i)$ by Lemma 3.

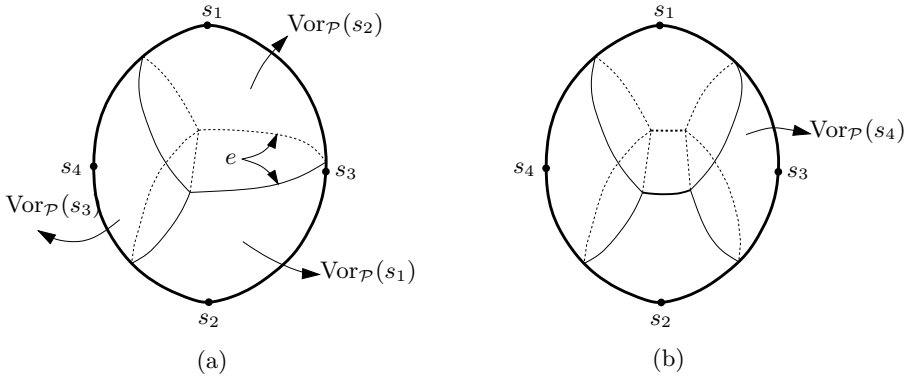


Fig. 2. (a) $G(N^3)$. No vertex of $G(N^3)$ is in conflict with s_4, but edge e conflicts with s_4. (b) $G(N^4)$.

The algorithm implements the three cases of Lemma 4. Consider the insertion of s_{i+1} into the data structure for $G(N^i)$. We first follow the conflict pointer of s_{i+1} to find the conflict $X(s_{i+1}) \in G(N^i)$ in constant time.

If $X(s_{i+1}) = \emptyset$, we have case (i) of the lemma, and nothing needs to be done.

If $X(s_{i+1})$ is an edge, no vertex of $G(N^i)$ is in conflict with s_{i+1}. We have therefore case (ii) of the lemma, and there is no other conflict of s_{i+1} at all. We

can update our data structure to represent $G(N^{i+1})$ by removing a portion of e and replacing it with an eye-like subgraph that consists of two edges induced by the two bisector planes between s_{i+1} and each of the two sites defining e in N^i. See Fig. 3. Updating the adjacency relations takes constant time. However, we also need to compute the coordinates of the two new vertices. By Lemma 2, this can be done by two ray shooting queries on $\mathcal{P}$ in time $O(\log m)$.

If $X(s_{i+1})$ is a vertex, we have case (iii) of the lemma. By Lemma 3, the portion of $G(N^i)$ lying inside $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ is a connected subgraph G of $G(N^i)$. We identify G by traversing $G(N^i)$. This takes time linear in the size of G, as we only need to test conflicts between vertices and s_{i+1}, which takes constant time per test. The extremal edges of G lie on the boundary of the new face $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. We shorten these edges by creating new vertices. After generating the new boundary by connecting these vertices, we finally delete G. If the complexity of G is k, all this can be done in time $O(k \log m)$ using Lemma 2. See Fig. 3.

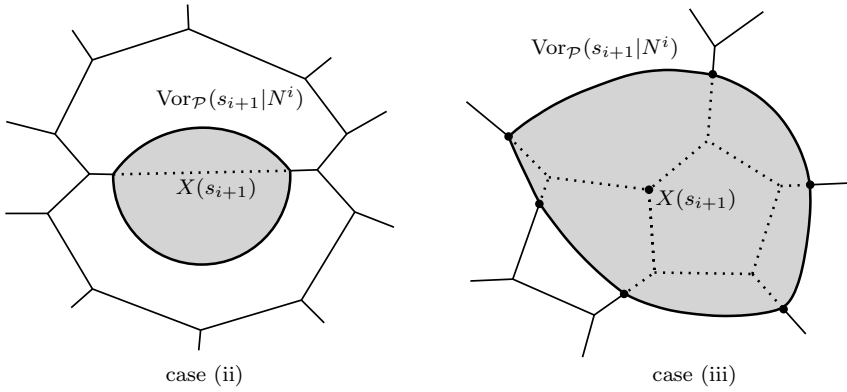


Fig. 3. Illustrating two cases of Lemma 4.

5 Conflict Maintenance

It remains to see how we update the conflict list during the insertion of a new site s_{i+1}. Recall that we maintain for every site $s \in N \setminus N^i$ a bidirectional pointer to a vertex or edge of $G(N^i)$ in conflict with s, if there is one. More precisely, if there are vertices of $G(N^i)$ in conflict with s, then $X(s)$ is one of them. If no vertex of $G(N^i)$ is in conflict with s and $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \neq \emptyset$, then there must be an edge $e \subset G(N^i)$ in conflict with s by Lemma 4 (ii). In this case, we set $X(s) = e$. Notice that if $X(s)$ is an edge, then no vertex in $G(N^i)$ conflicts with s (see Lemma 4 (ii)).

When inserting s_{i+1}, a new face $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ is defined. If $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \neq \emptyset$, then the vertices and edges in conflict with s_{i+1} will be destroyed in $G(N^{i+1})$. These are exactly the vertices and edges of $G(N^i)$ that intersect $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$.

Let s be a non-inserted site in $N \setminus N^{i+1}$. If $X(s)$ intersects $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, then $X(s)$ is not defined in $G(N^{i+1})$ any more, and we need to update $X(s)$ by finding a vertex or edge of $G(N^{i+1})$ in conflict with s. Otherwise, namely if $X(S)$ does not intersect $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, then we do not need to update $X(s)$. The next lemmas will be used for such conflict update. The boundary of a set R is denoted by ∂R.

Lemma 5. *Suppose that* $s \in N \setminus N^{i+1}$ *and* $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \neq \emptyset$. *If* $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \subset \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ *then* $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1}) = \emptyset$. *Otherwise* $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \subset \mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$.

Proof. If $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \subset \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, then all the vertices and edges of $G(N^i)$ lying inside $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ would be destroyed at the end of step $i+1$. It is equivalent to say that no vertex and no edge is in conflict with s over $G(N^{i+1})$. By Lemma 4 (i) it follows that $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$ is empty.

Let now $x \in \mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Since $x \in \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, there is some $s' \in N^i$ such that $d(x, s_{i+1}) = d(x, s')$. However, since $x \in \mathrm{Vor}_{\mathcal{P}}(s|N^i)$, we have $d(x, s) > d(x, s')$, and so $d(x, s) > d(x, s_{i+1})$. This implies $x \in \mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$.

A point p is said to be *visible* from a point q within $G(N^i)$ if there is a path connecting q to p whose interior does not intersect any vertex or edge of $G(N^i)$. Similarly, an edge e is said to be *visible* from a point q within $G(N^i)$ if there is a point on e that is visible from q.

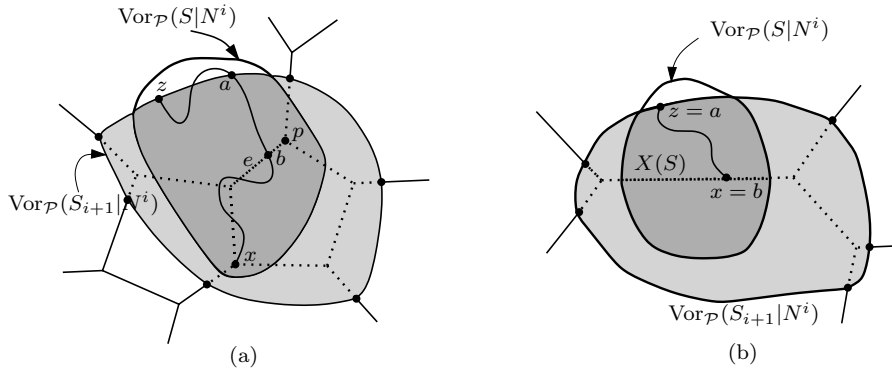


Fig. 4. The proof of Lemma 6 (ii). (a) $X(s)$ is a vertex. (b) $X(s)$ is a portion of an edge. In this example, $z = a$ and $x = b$.

Lemma 6. *Suppose that* s *is a non-inserted site in* $N \setminus N^{i+1}$ *and* $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$
$\neq \emptyset$. *If* $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1}) \neq \emptyset$, *then one of these two cases must occur:*

(i) s *has a conflicting vertex* $v \in \mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ *in* $G(N^{i+1})$.

(ii) *S has no conflicting vertex in* $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$*, but has a single conflicting edge* $e \in \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ *such that* e *is visible from a point* $p \in G(N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s|N^i)$ *within* $G(N^i)$.

Proof. By Lemma 5, each point of $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ is in conflict with s. Define $C = \mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Then C is a connected chain due to Lemma 1. If C contains a vertex of $G(N^{i+1}$, case (i) is true.

If C is a portion of an edge e of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, we need to prove that e is visible from some point $p \in G(N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s|N^i)$. For the following description, see Fig. 4.

Let z be a point of C. Let x be a point of $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Note here that if $X(s)$ is a vertex, then $x = X(s)$; if $X(s)$ is an edge, x is any point of $X(s)$ which belongs to $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Consider an arbitrary simple path γ connecting x to z within $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ (not within $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$). Since x lies in $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ and z does not lie in the interior of $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, γ must intersect $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ at least once, possibly in z. We denote by a the first such intersection on the way from x to z. Let e be the edge of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ containing a.

Let b be the last intersection of $G(N^i)$ with γ on the way from x to a. If $b = x$, then it means e is an edge bounding some region in $G(N^i)$ incident to x, so e is visible from x, that is, $p = x$. If $b \neq x$, Lemma 4 (iii) implies that an endpoint p of the edge of $G(N^i)$ containing b conflicts with s. It means that a is visible from p, so e is visible from p, and e intersects γ. Hence e conflicts with s in $G(N^{i+1})$.

Lemma 7. *For a site* $s \in N \setminus N^i$ *and a given vertex or edge of* $G(N^i)$*, we can decide in* $O(\log m)$ *time whether they are in conflict.*

Proof. A vertex $w \in G(N^i)$ is in conflict with s if and only if w is farther from s than the three sites in N^i that define w. This can be checked in constant time because we have access to the coordinates of w and the sites defining the adjacent faces.

Suppose now that we want to check whether an edge e is in conflict with a site s. Let $s', s'' \in N^i$ be the sites defining the faces adjacent to e, and let π be the bisecting plane of s', s''. The edge e is embedded in π, and is in conflict with s if and only if it intersects the halfplane $\pi' := \{x \in \pi \mid d(x,s) > d(x,s')\}$. We test this by ray shooting on $\mathcal{P}$ in time $O(\log m)$.

We are now ready to describe how the conflict list is updated during the insertion of a new site s_{i+1}. We first collect all not-yet-inserted sites s whose conflict is destroyed by the insertion of s_{i+1}. This can be done during the exploration of the subgraph. Since $X(s)$ is being deleted in $G(N^{i+1})$, we need to either find a new conflicting object in $G(N^{i+1})$, or to find out that there is none, which means that $\mathrm{Vor}_{\mathcal{P}}(s|N^j) = \emptyset$ for all $j \geq i+1$ by Lemma 5.

Suppose that $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$ is not empty. By Lemma 6, we can find a new vertex or edge in conflict with s by exploring the subgraph of $G(N^i)$ belonging

to $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ as follows. If $X(s)$ is an edge, it may have a conflict with a new vertex of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ that is defined in the interior of $X(s) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ or a new edge that is visible from any point in $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. In the latter case, there can be at most two such edges, thus we can check if the edges conflict with s. If they do not, then s can be discarded since its Voronoi region is empty.

On the other hand, if $X(s)$ is a vertex, then we first look for a new conflicting vertex by walking among the vertices of $G(N^i)$ in conflict with s until we reach a vertex of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. If we do not find such a vertex, we still have to check if there is a new conflicting edge of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ which is visible from some vertex visited so far (see Lemma 6 (ii)). If we fail to find a conflicting edge, then s can be discarded.

6 Analysis

Two of our primitive operations—creating a Voronoi vertex and detecting conflict with an edge—require ray shooting query on $\mathcal{P}$, and therefore take $O(\log m)$ time. All other primitive operations take constant time. After the initial preprocessing in time $O(m \log m)$, a backward analysis argument (see pp. 98–108 in [5]) shows that our algorithm performs an expected number of $O(n \log n)$ primitive operations. This implies a running time of $O(m \log m + n \log n \log m)$.

Theorem 2. *Given a polytope $\mathcal{P}$ with m vertices, and n points N on the surface of poly, we can compute $G(N) = G_{\mathcal{P}}(N)$ in expected time $O(m \log m + n \log n \log m)$.*

7 Point Location Queries and Farthest Neighbors

Once we have computed the implicit representation of the farthest-point Voronoi diagram of N on $\mathcal{P}$, we can easily find a point in the interior of each face $\mathrm{Vor}_{\mathcal{P}}(s|N)$. For example, take two non-adjacent vertices in an ordinary face, or the two vertices of an eye, and shoot a ray from s towards their midpoint. Then, run the same algorithm again, but now computing a radial triangulation of $G(N)$ (see pp. 109 in [5]). Each of its faces is the intersection of $\mathcal{P}$ with a cone (ss', sv, sv') where $s \in N$, s' is the interior point for $\mathrm{Vor}_{\mathcal{P}}(s|N)$, and v, v' are consecutive vertices of $\mathrm{Vor}_{\mathcal{P}}(s|N)$. During the second pass, we maintain for each site, whether it has been inserted or not, a pointer to the face of the radial triangulation that contains it. One can see easily that this does not hurt our time bound. It allows to find for each site the face of $G(N)$ it belongs to. Still following [5] we can build a point location data structure for our radial triangulation that answer queries in $O(\log^2 n)$ time with high probability. The reason why we need the radial triangulation for these two results is a configuration space argument, namely we want the faces of our graph to have bounded degree.

Theorem 3. *Given a set N of points in 3-dimensional convex positions, all-pairs farthest neighbors can be computed in expected $O(n \log^2 n)$ time. A data*

structure that answers farthest-neighbor queries in $O(\log^2 n)$ *time with high probablility can be build within the same time bounds.*

8 Conclusion and Open Problems

The main open problem is to fill the gap between the time complexity of our algorithm, namely $O(n \log^2 n)$, and the lower bound $\Omega(n)$. Besides, one can try to find a deterministic equivalent, but this looks hard. Indeed, a simple $O(n \log n)$ algorithm [4] for the diameter problem has been found in 1989 but a deterministic $O(n \log^2 n)$ [7] time algorithm was only published in 1997.

References

1. Pankaj K. Agarwal, J. Matoušek, and Subhash Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom. Theory Appl.*, 1(4):189–201, 1992.
2. A. Aggarwal and D. Kravets. A linear time algorithm for finding all farthest neighbors in a convex polygon. *Information Processing letters*, 31(1):17–20, 1989.
3. Sergei N. Bespamyatnikh. An efficient algorithm for the three-dimensional diameter problem. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 137–146, 1998.
4. K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
5. K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.
6. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.
7. E. Ramos. Construction of 1-d lower envelopes and applications. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pp. 57–66, 1997.
8. E. Ramos. An optimal deterministic algorithm for computing the diameter of a 3-d point set. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, page to appear, 2000.
9. P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

Finding an Optimal Bridge between Two Polygons

Xuehou Tan

Tokai University, 317 Nishino, Numazu 410-0395, Japan
tan@fc.u-tokai.ac.jp

Abstract. Let $\pi(a, b)$ denote the shortest path between two points a, b inside a simple polygon P, which totally lies in P. The *geodesic distance* between a and b in P is defined as the length of $\pi(a, b)$, denoted by $gd(a, b)$, in contrast with the Euclidean distance between a and b, denoted by $d(a, b)$. Given two disjoint polygons P and Q in the plane, the *bridge problem* asks for a line segment (*optimal bridge*) that connects a point p on the boundary of P and a point q on the boundary of Q such that the sum of three distances $gd(p', p)$, $d(p, q)$ and $gd(q, q')$, with any $p' \in P$ and any $q' \in Q$, is minimized. We present an $O(n \log^3 n)$ time algorithm for finding an optimal bridge between two simple polygons. This significantly improves upon the previous $O(n^2)$ time bound.

1 Introduction

The *bridge problem*, introduced by Cai *et al.* [2], is defined as: Given two disjoint regions, we want to build a bridge to connect them so as to minimize the length of the longest path between two points in different regions. Because of its similarity to many geometric problems (e.g., diameter problems, minimum separation problems and minimum spanning tree problems), much progress has been made, immediately after the problem is posed. For the simple case where two polygons are convex, two optimal algorithms have been presented by Kim and Shin [5] and by Tan [8], independently. Kim and Shin also reported an $O(n^2)$ time algorithm for finding an optimal bridge between two simple polygons [5]. A linear time solution to the case of two rectilinear polygons is given in [10]. Efficient algorithms for high-dimensional bridge problems can be found in [8,9].

In this paper, we present an $O(n \log^3 n)$ time algorithm for finding an optimal bridge between two simple polygons with a total of n vertices. Our result is obtained by making use of a hierarchical structure that consists of the well-known *segment trees*, *range trees* and *persistent search trees* [6,7], and a structure that supports dynamic ray shooting and shortest path queries as well [3].

2 Necessary Conditions for Optimal Bridges

Let $\pi(a, b)$ denote the shortest path between two points a, b inside a simple polygon P, which starts at a, ends at b and totally lies in P. The length of

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 170–180, 2001.
© Springer-Verlag Berlin Heidelberg 2001

$\pi(a, b)$ is the *geodesic distance* between a and b in P, and is denoted by $gd(a, b)$. Usually, the Euclidean distance between a and b is denoted by $d(a, b)$.

Let P and Q denote two disjoint polygons in the plane with boundaries ∂P and ∂Q, respectively. The *bridge problem* considered in [2] deals with finding a line segment that connects a point $p \in \partial P$ and a point $q \in \partial Q$ so that the length function $max_{p' \in P}\{gd(p', p)\} + d(p, q) + max_{q' \in Q}\{gd(q, q')\}$ is minimized. We call the segment (p, q) the *optimal bridge* between P and Q.

We will first review necessary conditions for optimal bridges between two convex polygons given in [8], and then generalize them to simple polygons.

2.1 The Case of Two Convex Polygons

Notice that in the case of convex polygons, $gd(p', p) = d(p', p)$ and $gd(q, q') = d(q, q')$. Let $V(P)$ denote the furthest-site Voronoi diagram of vertices of polygon P and $V(p_i)$ the furthest-site Voronoi region of a vertex p_i [6]. Let $p_{i,j}$ denote the intersection point of ∂P with the common edge of regions $V(p_i)$ and $V(p_j)$. Define the infinite triangular region, bounded by two half lines being collinear with but not containing the segments $(p_i, p_{i,j})$ and $(p_j, p_{i,j})$, as the *fan* of $p_{i,j}$, and denote it by $F(p_{i,j})$ (see Fig. 1b). The fan $F(p_{i,j})$ specifies the area to which the candidate bridges with $p = p_{i,j}$ belong (see Lemma 1). Also, let p_{i1} and p_{i2} be two intersection points of ∂P with the boundary of a region $V(p_i)$. Define the *fan* of the vertex p_i, denoted by $F(p_i)$, as the triangular region bounded by two half lines passing through and containing the segments (p_i, p_{i1}) and (p_i, p_{i2}). The fan $F(p_i)$ specifies the area to which the candidate bridges with $p' = p_i$ belong. The same definition also applies to polygon Q.

Lemma 1 *The points p and p' of an optimal solution must satisfy one of the following conditions (the same conditions symmetrically hold for the points q and q'):*

(1) p is the intersection point of ∂P with the common edge of $V(p_i)$ and $V(p_j)$, and the fan $F(p)$ contains the other point q of the optimal bridge, or
(2) p is the intersection point of ∂P with the segment (p', q) and the fan $F(p')$ contains the point q, where q satisfies the condition (1), or
(3) p is the intersection point of ∂P with the segment (p', q'), and the fans $F(p')$ and $F(q')$ contain the vertices q' and p', respectively.

Proof. We will give a simple proof of the condition (1) and omit the proofs of conditions (2) and (3) (see also [8]).

Consider first the case where any two of segments (p', p), (p, q) and (q, q') of an optimal solution are not collinear. Assume that vertex p_i is the point p' of the optimal solution. Obviously, a contradiction occurs if p_i is not the vertex farthest to the point p of the optimal bridge in polygon P. If p_i is the only vertex farthest to the point p in P (Fig. 1a), or p_i and another vertex p_j are the vertices farthest to the point p $(= p_{i,j})$ but the point q of the optimal bridge is not contained in the fan $F(p_{i,j})$ (Fig. 1b), we can move p by a very small distance to obtain a new point p^* such that p^* and q are mutually visible, p_i or

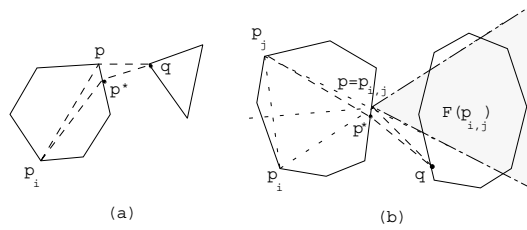


Fig. 1. Illustration for the proof of Lemma 1.

p_j is still the vertex farthest to p^*, and $d(p_i, p^*) + d(p^*, q) < d(p_i, p) + d(p, q)$ or $d(p_j, p^*) + d(p^*, q) < d(p_j, p) + d(p, q)$, a contradiction. Therefore, the points p and p' of the optimal solution satisfy the condition (1). □

2.2 The Case of Two Simple Polygons

Given a simple polygon P' of m vertices and a set $S = \{s_1, \ldots, s_k\}$ of k sites inside P', the *farthest-site geodesic Voronoi diagram* of S in P' is the partition of P' into k regions $GV(s_1), \ldots, GV(s_k)$ (some of them may be empty) such that a point $x \in P'$ belongs to $GV(s_i)$ if and only if the geodesic distance of site s_i to x is farther than that of any other site to x. An $O((m+k)\log(m+k))$ time algorithm has been presented to compute the farthest-site geodesic Voronoi diagram of k sites in a simple polygon of m vertices [1].

Let $GV(P)$ be the farthest-site geodesic Voronoi diagram of vertices of the simple polygon P. Let p_i, p_j denote two vertices of P such that their regions, $GV(p_i)$ and $GV(p_j)$, are adjacent, and $p_{i,j}$ the intersection point of ∂P with the common edge of $GV(p_i)$ and $GV(p_j)$. Similarly, the fan $F(p_{i,j})$ needs to be defined. Let L_1 and L_2 be two infinite rays shot at $p_{i,j}$, being collinear with but not containing the first segments of two paths $\pi(p_{i,j}, p_i)$ and $\pi(p_{i,j}, p_j)$, respectively, and let $T(L_1, L_2)$ denote the triangular region bounded by L_1 and L_2 (see Fig. 6). Since parts of polygon P may appear in $T(L_1, L_2)$, i.e, they block the visibility of the point $p_{i,j}$, region $T(L_1, L_2)$ cannot simply be defined as the fan $F(p_{i,j})$. Let $[p^1_{i,j}, p^2_{i,j}]$ be the *maximal* interval on ∂Q such that $p^1_{i,j}$ and $p^2_{i,j}$ lie in $T(L_1, L_2)$ and both of them are visible from $p_{i,j}$. We then define the fan $F(p_{i,j})$ as the triangular region bounded by two half lines being collinear with and containing the segments $(p_{i,j}, p^1_{i,j})$ and $(p_{i,j}, p^2_{i,j})$.

For a non-empty region $GV(p_i)$, we further divide the boundary of $GV(p_i)$, overlapped with ∂P, into the intervals such that all points within an interval have (or are visible from) the same first turn point in their shortest paths to p_i. See Fig. 2 for an example. Note that a first turn point may have several intervals. In [1], Aronov *et al.* referred to such a first turn point as the *anchor* of all points of the interval with respect to p_i, and defined a *refined Voronoi diagram* so that all points in a refined region have the same furthest vertex and the same anchor with respect to the furthest vertex.

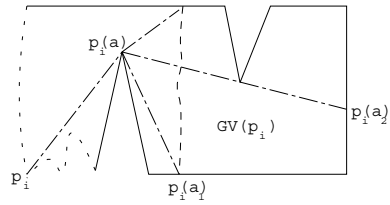


Fig. 2. First turn points (anchors) with their intervals.

Let $p_i(a)$ be an anchor with respect to p_i, and $[p_i(a_1), p_i(a_2)]$ one of its interval(s). See Fig. 2. To define the fan $F(p_i(a))$ with respect to the interval $[p_i(a_1), p_i(a_2)]$, we modify ∂P by removing the interval $[p_i(a_1), p_i(a_2)]$, and by inserting the edges $(p_i(a), p_i(a_1))$ and $(p_i(a), p_i(a_2))$. The definition of $F(p_i(a))$ is then similar to that of $F(p_{i,j})$, with a slight modification that $T(L_1, L_2)$ is replaced by the region bounded by two half lines being collinear with and containing $(p_i(a), p_i(a_1))$ and $(p_i(a), p_i(a_2))$. (An anchor has as many fans as the intervals it has.) The same definitions also apply to polygon Q.

Lemma 2 *Let $p'(a)$ and $q'(a)$ denote the anchors of p and q with respect to p' and q', respectively. The points p and p' of an optimal solution must satisfy one of the following conditions (the same conditions symmetrically hold for the points q and q'):*

(1) p is the intersection point of ∂P with the common edge of $GV(p_i)$ and $GV(p_j)$, and $F(p)$ contains the other point q of the optimal bridge, or
(2) p is the intersection point of ∂P with the segment $(p'(a), q)$ and $F(p'(a))$ contains the point q, where q satisfies the condition (1), or
(3) p is the intersection point of ∂P with the segment $(p'(a), q'(a))$, and $F(p'(a))$ and $F(q'(a))$ contain the anchors $q'(a)$ and $p'(a)$ respectively, or
(4) (p, q) is a line segment that touches a vertex v of P or Q, and three points $p'(a)$, $q'(a)$ and v are in the same side of the line passing through (p, q).

Proof. Consider first the case where the optimal bridge (p, q) is not collinear to the first segment of the path $\pi(p, p')$ or $\pi(q, q')$. Assume that vertex p_i is the point p' of the optimal solution. In the case where p_i is the only vertex of P farthest to the point p in the geodesic metric, or p_i and another vertex p_j are the vertices farthest to the point p $(= p_{i,j})$ but the point q of the optimal bridge is not contained in the fan $F(p_{i,j})$, a contradiction occurs, provided that point p can be moved by a small distance to get a smaller value of $gd(p', p) + d(p, q) + gd(q, q')$. The condition (1) is satisfied in this case.

The conditions (2) and (3) can be proved by an argument similar to the proof of Lemma 1 (see [8] for a full proof), with a slight modification that p is the intersection point of ∂P with the segment $(p'(a), q)$ or $(p'(a), q'(a))$.

Since polygons P and Q are simple, the bridge (p, q) may touch a vertex v of P or Q. See Fig. 3. (In this case, point p cannot simply be moved to give a contradiction.) Without loss of generality, assume that v is a vertex of P. Let p_i

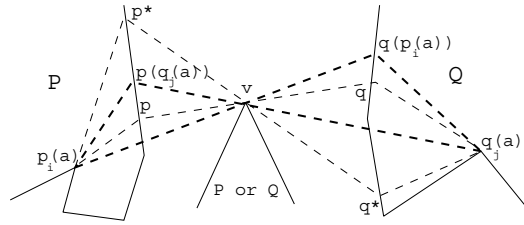


Fig. 3. A simple case in the rotation of the bridge (p, q).

(resp. q_j) denote the vertex of P (resp. Q) farthest to the point p (resp. q) in the geodesic metric, $p_i(a)$ (resp. $q_j(a)$) the anchor of p (resp. q) with respect to p_i (resp. q_j), and $q(p_i(a))$ (resp. $p(q_j(a))$) the intersection point of ∂Q (resp. ∂P) with the line passing through $(v, p_i(a))$ (resp. $(v, q_j(a))$). See Fig. 3 for an example. Obviously, the optimal bridge cannot be located at the position of (p^*, q^*), where p^* (or q^*) is not contained in the triangle determined by $p_i(a)$, $p(q_j(a))$ (or $q_j(a)$, $q(p_i(a))$) and v; otherwise, $d(p_i(a), p(q_j(a))) + d(p(q_j(a)), q_j(a)) < d(p_i(a), p^*) + d(p^*, q^*) + d(q^*, q_j(a))$, a contradiction (Fig. 3). Consider a process of rotating the bridge (p, q) around point v, with the point p contained in the triangle described above. Assume first that nothing is encountered in the process of rotating the bridge (p, q), i.e., it begins with $p = p(q_j(a))$ and ends with $q = q(p_i(a))$. The sum $d(p_i(a), p) + d(p, v)$ is monotone decreasing in the rotating process, while the sum $d(v, q) + d(q, q_j(a))$ is monotone increasing. Therefore, the length function $d(p_i(a), p) + d(p, v) + d(v, q) + d(q, q_j(a))$ as well as $gd(p', p) + d(p, q) + gd(q, q')$ is convex with a unique local minimum. If the minimum occurs at point $p^\star$ in the rotating process, then $(p^\star, q^\star)$ is a candidate bridge. This gives the condition (4).

In the rotating process, the bridge (p, q) may touch two vertices of P or/and Q, and the vertex v of polygon P or the anchor $q_j(a)$ may be changed. Consider first the situation where the endpoint q of the current bridge reaches the endpoint of $q_j(a)$'s interval on ∂Q (Fig. 4a). In this case, the anchor of q_j is changed to a new one, say, $q'_j(a)$ in Fig. 4a. (The vertex q_j needn't be changed in the rotating

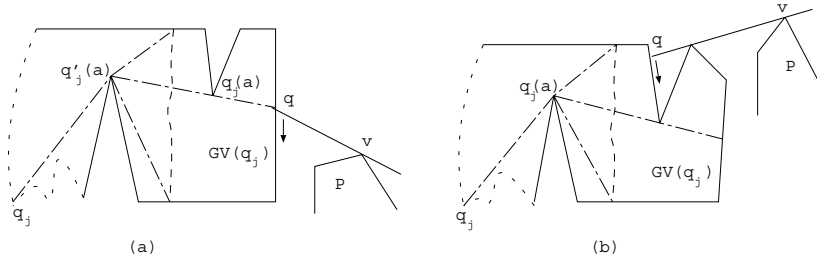


Fig. 4. Two more cases in the rotation of the bridge (p, q).

process; otherwise, point q were not contained in the triangle described above.) In the situation where the current bridge (p, q) makes a tangent contact with a portion (or two edges) of Q, the candidate bridge should be found before or at the tangent contact (Fig. 4b). In the situation where the current bridge (p, q) touches another vertex v' of polygon P, the center of the rotation is changed from v to v'. In all cases, the rotated ranges on both ∂Q and ∂P are continuous Therefore, the length function considered in the process of rotating the bridge (p, q) is convex, and thus we obtain the condition (4). □

3 The Algorithm

Two types of the fans defined in Section 3.2 are dealt with by the following two lemmas, respectively.

Lemma 3 *Let p_i, p_j denote two vertices of P such that $GV(p_i)$ and $GV(p_j)$ are adjacent, and $p_{i,j}$ the intersection point of ∂P with their common edge. All fans $F(p_{i,j})$, for $p_i, p_j \in P$, can be computed in $O(n \log^2 n)$ time.*

Proof. Before describing our algorithm, we briefly review the result of Goodrich and Tamassia on dynamic ray shooting and shortest path queries [3]. For a *connected subdivision* of the plane into simple polygons (i.e., no "islands" exist), one can maintain a data structure of size $O(n)$ that supports query time of $O(\log^2 n)$ for a ray shooting or shortest path in a region of the subdivision (plus $O(k)$ if the path of length k is to be reported), and update time of $O(\log^2 n)$ for each addition/deletion of an edge/vertex.

To use dynamic ray shooting and shortest path algorithms, we construct a planar subdivision that always consists of a bounded region and an unbounded region. The bounded region, denoted by PQ, is an approximation of the exterior of the union $P \cup Q$. The region PQ is initially constructed as follows. Let a be the highest point among all vertices of P and Q. Place two points b and c sufficient below and to the left and right of the union of P and Q, respectively, and connect three points a, b, c so that P and Q are contained in the triangle $\triangle_{abc}$. Make a copy a' of the vertex a. Then, find a point p^* on ∂P and a point q^* on ∂Q such that they are visible each other. Split each of p^* and q^* into two and insert two edges: one connects vertices p^* and q^*, and the other does their copies. The resulting (simple) polygon PQ turns the original polygons P and Q inside out (Fig. 5).

It is easy to see that a ray shooting query in the exterior of the union $P \cup Q$ can be performed in PQ with a slight modification that the ray hits nothing if the hit point is on the boundary of $\triangle_{abc}$, and an extra ray shooting needs to perform if the hit point is on the *pseudo-edge* (p^*, q^*) or its copy. Special attention should be paid to the shortest path queries. This is because a shortest path in the exterior of the union $P \cup Q$ may require to go across the pseudo-edges or even wrap around the highest vertex a. When a shortest path query is performed in PQ, we first find a new pair of mutually visible points, one on ∂P and the other on ∂Q, such that the shortest path to be computed should not go

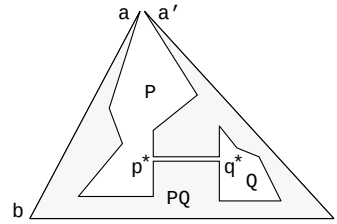


Fig. 5. Turning polygons P and Q inside out.

across the line segment connecting them. After the polygon PQ is accordingly modified, the shortest path query can safely be performed. Also, the shortest path between two points u and v in the exterior of $P \cup Q$ is the shortest one of three paths $\pi(u, v)$, $\pi(u, a) \cup \pi(a, v)$ and $\pi(u, a') \cup \pi(a', v)$ in PQ.

First, all points $p_{i,j}$ can be obtained from the intersections of ∂P with edges of the diagram $GV(P)$. To compute the interval $[p^1_{i,j}, p^2_{i,j}]$ on ∂Q, we shoot two rays at $p_{i,j}$ along L_1 and L_2 in polygon PQ. If both of them hit ∂Q, the interval $[p^1_{i,j}, p^2_{i,j}]$ is obtained and we are done. If only one of them, say, the point hit by the ray along L_2, is on ∂Q, then we obtain the point $p^2_{i,j}$. To compute the point $p^1_{i,j}$, we modify polygon PQ by inserting the edge connecting $p_{i,j}$ with the hit point on ∂Q and its copy, and by removing the old pseudo-edges. Let f_2 denote the intersection point of ∂Q with L_2, which is farthest to $p_{i,j}$ along L_2. Point f_2 can be computed by shooting a ray at infinity in the converse direction of L_2 in the exterior of polygon Q. See Figs. 6a-b. Then, find the first turn point of the path $\pi(p_{i,j}, f_2)$ in the polygon PQ. If the turn point is on ∂Q, it is just $p^1_{i,j}$ (Fig. 6a). Otherwise, shoot a ray at $p_{i,j}$ to the first turn point (on ∂P), and the hit point on ∂Q gives $p^1_{i,j}$ (Fig. 6b).

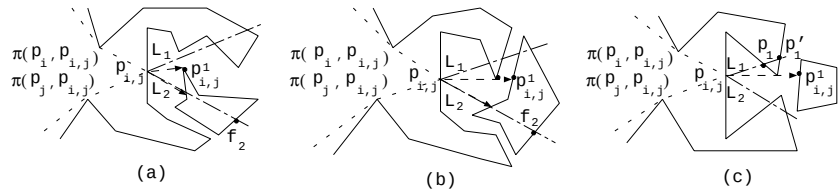


Fig. 6. Illustration for the proof of Lemma 3.

Consider now the situation where two rays along L_1 and L_2 hit ∂P. Let p_1 and p'_1 denote two intersection points of ∂P with L_1, closest and farthest to $p_{i,j}$, respectively. Modify the polygon PQ by inserting the segment $(p_{i,j}, p_1)$ and its copy, and by removing the old pseudo-edges. Then, find the first turn point of the path $\pi(p_{i,j}, p'_1)$ in the resulting polygon PQ. If the turn point is on ∂Q, it is an endpoint of the interval $[p^1_{i,j}, p^2_{i,j}]$. Otherwise, shoot a ray at $p_{i,j}$ towards that

turn point. If the hit point is on ∂Q, it gives an endpoint of $[p^1_{i,j}, p^2_{i,j}]$, say, $p^1_{i,j}$ in Fig. 6c; otherwise, the fan $F(p_{i,j})$ is empty. If $F(p_{i,j})$ is not empty, the other endpoint of $[p^1_{i,j}, p^2_{i,j}]$ can similarly be found by considering two intersections of ∂P with L_2, one closest to $p_{i,j}$ and the other farthest to $p_{i,j}$.

Finally, consider the situation where two rays shot at $p_{i,j}$ along L_1 and L_2 hit nothing. If a point of Q is contained in $T(L_1, L_2)$, the convex hull $CH(Q)$ of Q is completely contained in $T(L_1, L_2)$, and thus points $p^1_{i,j}$, $p^2_{i,j}$ can be obtained from two tangent points of $p_{i,j}$ to $CH(Q)$. Otherwise, the fan $F(p_{i,j})$ is empty.

Since a constant number of operations is needed, the interval $[p^1_{i,j}, p^2_{i,j}]$ can be computed in $O(\log^2 n)$ time. Hence, the lemma follows. □

Lemma 4 *Let $GV(p_i)$ be a non-empty farthest-site geodesic Voronoi region, and $A(p_i)$ the set of anchors of all points on the boundary of $GV(p_i)$ with respect to vertex p_i. All fans $F(p_i(a))$, for $p_i \in P$ and $p_i(a) \in A(p_i)$, can be computed in $O(n \log^2 n)$ time.*

Proof. As it can be proved by an argument similar to the proof of Lemma 3, we omit the detail in this extended abstract. □

By now, we can give the main result of this paper using the results obtained in the previous sections.

Theorem 1 *An optimal bridge between two simple polygons P and Q can be computed in $O(n \log^3 n)$ time.*

Proof. First, compute the furthest-site geodesic Voronoi diagrams of vertices of P and Q, respectively [1]. As shown in Lemmas 3 and 4, all fans of P and Q can be computed in $O(n \log^2 n)$ time.

It follows from the condition (1) of Lemma 2 that if p_i and p_j are two vertices farthest to the point $p_{i,j}$ in the geodesic metric, then $p_{i,j}$ is a candidate of the points p of optimal bridges. In particular, we call such points $p_{i,j}$ the *candidate points*. For convenience, we denote by $p^1, \ldots, p^l$ and $q^1, \ldots, p^m$ the sequences of candidate points on ∂P and ∂Q, respectively, and $w(p^i)$ ($w(q^j)$) the geodesic distance from p^i (q^j) to the vertex farthest to p^i in P (q^j in Q).

To quickly find the candidate bridges satisfying the first three conditions of Lemma 2, we construct a hierarchical structure that consists of segment trees, range trees and persistent search trees [6,7]. For this purpose, we need a new idea to represent the fan of a candidate point. Order all vertices and candidate points on ∂P (∂Q), say, by a clockwise scan of the polygon boundary, starting at a specified vertex $v_p \in P$ ($v_q \in Q$). For a complete ordering, we consider the point v_p as two points v_{pl} and v_{pr} such that $v_{pl} \leq p \leq v_{pr}$, for all points p on ∂P. The treatment for the point v_q is the same. Let p^{i1}, p^{i2} denote the first intersection points of ∂Q with two bounding half lines of $F(p^i)$. The fan $F(p^i)$ can then be represented by the interval $[p^{i1}, p^{i2}]$ if v_q is not contained in $F(p^i)$, or two intervals $[v_{ql}, p^{i1}]$ and $[p^{i2}, v_{qr}]$ otherwise.

Consider now how to compute an optimal bridge (p, q) that is not collinear to the first segment of the path $\pi(p, p')$ or $\pi(q, q')$. For each point p^i $(1 \leq i \leq l)$, we want to find a point q^j such that p^i and q^j lie in $F(q^j)$ and $F(p^i)$, respectively,

and $d(p^i, q^j) + w(q^j)$ is minimized. (In this case, $w(p^i) + d(p^i, q^j) + w(q^j)$ is also minimized.) First, organize the intervals of all fans $F(q^j)$ $(1 \leq q^j \leq m)$ into a segment tree. An interval is stored in $O(\log n)$ nodes, and thus each node s of the segment tree has an associated set $COVER(s)$ of the intervals. For each $COVER(s)$, store the candidate points whose corresponding intervals are in $COVER(s)$ into a second-level range tree. Again, a candidate point is stored in $O(\log n)$ nodes, and each node r of the range tree for $COVER(s)$ has an associated set $SUB(r)$ of candidate points. For each $SUB(r)$, we further compute the additively weighted (nearest) Voronoi diagram with the weights $w(q^j)$ for all candidate points q^j in $SUB(r)$, and organize the diagram into a third-level persistent search tree. The resulting data structure is called the *segment-range-persistent tree*. Since it takes $O(n \log n)$ time and space to construct a segment or range tree, and $O(n \log n)$ time and $O(n)$ space to compute a Voronoi diagram and a persistent search tree, the segment-range-persistent tree can be constructed in $O(n \log^3 n)$ time and $O(n \log^2 n)$ space [6,7].

Given a query point p^i, we find the point q^j which is optimal with respect to p^i as follows. In the top-level segment tree, find the sets $COVER(s)$ whose intervals contain p^i. In the second-level range trees storing the found sets $COVER(s)$, we further find the sets $SUB(r)$ whose candidate points (on ∂Q) are contained in the interval $[p^{i1}, p^{i2}]$ or in two intervals $[v_{ql}, p^{i1}]$ and $[p^{i2}, v_{qr}]$. In the third-level persistent search trees, find the candidate points of Q which are locally nearest to p^i using the point location algorithm [7]. The point q^j optimal with respect to p^i is then found from answers of these point location queries. Clearly, the query time taken for p^i is $O(\log^3 n)$. Since the number of candidate points on ∂P is $O(n)$, the time taken to enumerate all of such candidate bridges is $O(n \log^3 n)$.

Analogously, the candidate bridges satisfying the conditions (2) and 3 of Lemma 2 can be computed in $O(n \log^3 n)$ time. We omit the detail in this extended abstract.

To find the candidate bridges satisfying the condition (4), we consider the bounding half lines of anchor's fans which touch a vertex of P or Q. Assume that a bounding line of the fan $F(p_i(a))$ touches a vertex v of P and that $q(p_i(a))$ is the first intersection point of the bounding line with ∂Q. Let q_j denote the vertex of Q furthest to $q(p_i(a))$ in the geodesic metric, and $q_j(a)$ the anchor of $q(p_i(a))$ with respect to q_j. If $q_j(a)$ and v are in the same side of the line through $(p_i(a), q(p_i(a)))$, we should perform the rotating process described in the proof of Lemma 2.

Assume that p_1 is the intersection point of ∂P with the segment $(p_i(a), v)$, and p_2 the endpoint of the edge of P which contains p_1 and is contained in $F(p_i(a))$. Consider the process of rotating the bridge (p, q), with point p contained in the edge (p_1, p_2). First, we discuss how to handle the events (places) where rotation centers are changed. To find the first place where the center is changed, we compute the first turn point of the path $\pi(v, q_j(a))$ in the exterior of P, and the intersection point of the edge (p_1, p_2) with the line passing through v and the first turn point. (The first turn point of $\pi(p_1, q_j(a))$ in the exterior of P can be found in logarithmic time [4].) If the intersection point does not exist,

the center is kept unchanged in the process of rotating bridge (p, q), with point p contained in edge (p_1, p_2). If the intersection point is found, the rotation center is changed from v to that turn point when p reaches the intersection point. In this case, we consider (p_1, p_2) as two edges, separated by that intersection point. This allows us to consider only the situation where the center is kept unchanged in the process of rotating bridge (p, q), with point p contained in edge (p_1, p_2).

To find the candidate bridge quickly, we perform a binary search on the edges of Q which are rotated, using the distance computed for edge (p_1, p_2) and the chosen edge of ∂Q. (Given $p_i(a)$, v, $q_j(a)$ and a pair of the edges of P and Q, the position of p and q on the edges giving the local minimum can be computed in constant time.) Then, what is the range of the rotated edges on ∂Q? Clearly, the point $q(p_i(a))$ is the starting point of the rotated range. However, it takes $O(n)$ time to exactly give the ending point, as it may require to follow a sequence of anchors $q_j(a)$. To overcome this difficulty, we temporarily consider it as the endpoint of $GV(q_j)$ towards which the point q moves. (Recall that point q_j cannot be changed in the rotating process.) We then perform the binary search on the edges of Q contained in the range described above. Whenever the points p, q giving the local minimum do not exist in two considered edges, they occur like the segment (p^*, q^*) shown in Fig. 3, or the point of ∂Q hit by the ray shot at v towards q in the exterior of Q is not q (i.e., the segment (v, q) intersects with ∂Q), we consider the computed distance as ∞. This makes sure that the binary search should report a pair of points p and q giving the local minimum, with the point q contained in the real range of the rotated edges on ∂Q. If the reported point p is not identical to p_2, then (p, q) gives a candidate bridge. Otherwise, we need to perform the same procedure for the edge of ∂P, which is next to (p_1, p_2) and contained in $F(p_i(a))$. For this edge, the starting endpoint of the rotated range on ∂Q is already known. In this way, we can find all candidate bridges satisfying the condition (4) in $O(n \log n)$ time.

Finally, we simply report the one among all found bridges which minimizes the maximum distance between P and Q. It completes the proof. □

References

1. B.Aronov, S.Fortune and G.Wilfong, The furthest-site geodesic Voronoi diagram, *Discrete Comput. Geom.*, **9** (1993) 217-255.
2. L.Cai, Y.Xu and B.Zhu, Computing the optimal bridge between two convex polygons, *Inform. Process. Lett.* **69** (1999) 127-130.
3. M.T.Goodrich and R.Tamassia, Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations, *J. Algorithms* **23** (1997) 51-83.
4. L.J.Guibas and J.Hershberger, Optimal shortest path queries in a simple polygon, *J. Comput. Sys. Sci.* **39** (1989) 126-152.
5. S.K.Kim and C.S.Shin, *Computing the optimal bridge between two polygons*, HKUST Research Report TCSC-99-14, 1999.
6. F.Preparata and M.Shamos, *Computational Geometry*, Springer, Berlin, 1985.
7. N.Sarnak and R.E.Tarjan, Planar point location using persistent search trees, *Comm. ACM* **29** (1986) 669-679.

8. X.Tan, On optimal bridge between two convex regions, *Inform. Process. Lett.* **76** (2000) 163-168.
9. T.Tokuyama, Efficient algorithm for the minimum diameter bridge problem, Presented in *Japan Conference on Discrete and Computational Geometry'2000.*
10. D.P.Wang, An optimal algorithm for constructing an optimal bridge between two simple rectilinear polygons, To appear in *Inform. Process. Lett.*

How Good Is Sink Insertion?

Xiang-Yang Li and Yu Wang

Dept. of Computer Science, Illinois Institute of Technology,
10 W. 31st Street, Chicago, IL 60616.

Abstract. Generating high quality meshes is one of the most important steps in many applications such as scientific computing. Sink insertion method is one of the mesh quality improvement methods that had been proposed recently. However, it is unknown whether this method is competitive to generate meshes with small number of elements. In this paper, we show that, given a two-dimensional polygonal domain with no small angles, the sink insertion method generates a well-shaped mesh with $O(n)$ triangles, where n is the minimum number of triangles generated by any method with the same quality guarantee.
We also show that the sink insertion method more likely can not guarantee the same result for a three-dimensional domain, while the other methods such as Delaunay refinement can achieve.
Keywords: Mesh generation, Delaunay triangulations, sink insertion, computational geometry, algorithms.

1 Introduction

Mesh generation is the process of breaking a geometric domain into a collection of primitive elements such as triangles in 2D and tetrahedra in 3D. It has plenty of applications in scientific computing, computer graphics, computer vision, geometric information system, and medical imaging. Some applications have a strict quality requirement on the underlying meshes used. For example, most of the numerical simulations require that the mesh is well-shaped. In addition, some numerical simulations methods, for example, the control volume method, prefer the mesh to be a Delaunay triangulation.

Recently, Edelsbrunner *et. al* [2,3] proposed a new mesh improvement method based on sink insertion. This new method guarantees to generate a Delaunay mesh with a small radius-edge ratio. Edelsbrunner and Guoy (private communication) found that the sink insertion method tends to be more economical when we want to add as many points as possible at the same time to refine the mesh while maintaining the Delaunay property. It will also be useful in the software environment with off-line Delaunay triangulation or parallel meshing. In stead of dealing with all the circumcenters as many as the number of bad elements, they deal with small number of sinks of these bad elements. From experiments, they observe as many as 100 bad tetrahedra sharing the same sink. However, unlike Delaunay refinement, it is an open problem whether the sink insertion method generates an almost-good Delaunay mesh with $O(n)$ simplex elements,

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 181–190, 2001.
© Springer-Verlag Berlin Heidelberg 2001

where n is the minimum number of d-dimensional simplex elements generated by any other methods with the same radius-edge ratio quality guarantee.

In this paper, we show that the sink insertion method guarantees to generate a well-shaped mesh with size $O(n)$ in $2D$. For a three-dimensional domain, unlike the Delaunay refinement methods, the size optimality is not guaranteed because of the existence of slivers in an almost-good Delaunay mesh. We give an example that suggests that the sink insertion method may not guarantee to generate a mesh with size $O(n)$ for a three-dimensional domain.

The rest of the paper is organized as follows. In Section 2, we review the sink insertion algorithm proposed by Edelsbrunner *et. al.* In Section 3, we prove that the sink insertion method guarantees to generate a well-shaped mesh with size $O(n)$ for a two-dimensional PLC domain with no small angles. In section 4, we discuss why the sink insertion method may not guarantee that the generated mesh has size $O(n)$ for a three-dimensional PLC domain. Section 5 concludes the paper with further discussions.

2 Preliminary

A simplicial mesh is called *almost good* if each of its simplex elements has a small radius-edge ratio, which is the circumradius divided by the shortest edge length of the simplex. Hereafter we use $\rho(\tau)$ to denote the radius-edge ratio of a simplex τ. Several theoretical and practical approaches have been proposed for generating almost-good Delaunay meshes. Assume the spatial domain that does not have small angles is given in terms of its piecewise linear complex boundary (PLC) [7]. It has been shown that the Delaunay refinement methods [1,5,6] generate an almost-good Delaunay mesh with size $O(n)$, where n is the minimum number of elements for any mesh with the same radius-edge ratio for the same geometric domain.

2.1 Sink

Edelsbrunner *et al.* [2,3] defined the sink of a simplex σ in a Delaunay complex by the following recursive approach.

Definition 1. [SINK] *In a d-dimensional Delaunay complex, let c_σ be the circumcenter of a d-simplex σ; let $\mathcal{N}(\sigma)$ be the set of d-simplices that share a $d-1$ dimensional face with σ. For each simplex $\tau \in \mathcal{N}(\sigma)$, let H_τ be the half space containing τ bounded by the $d-1$ dimensional face shared by τ and σ. A point z is a* sink *of σ when*

- *z is c_σ and it is contained in σ; or*
- *z is a sink of $\tau \in \mathcal{N}(\sigma)$ and c_σ is contained in H_τ.*

A simplex containing its own circumcenter is called a *sink simplex*. Edelsbrunner *et al.* [2,3] showed that there is no loop in the definition of sink among all d-dimensional simplices by proving that the circumradius of the simplex τ

containing the sink of a simplex σ is not less than the circumradius of σ. Notice that if the circumcenter of a boundary simplex is not inside the domain, then its sink is not defined by the above definition. For this case, we just define its circumcenter as its sink. Given a boundary k-simplex σ ($k < d$) contained in a k-dimensional boundary polyhedron, its sink is defined by considering only that boundary polyhedron.

The *min-circumsphere* of a k-simplex τ in d-dimensions is the smallest d-dimensional sphere that contains all vertices of τ on its surface. When $k = d$, the min-circumsphere is also called the *circumsphere*. A point is said to *encroach* the domain boundary if it is contained inside the min-circumsphere of a boundary k-simplex, where $k < d$.

Let T be the set of tetrahedron in a 3-dimensional Delaunay mesh and $\mathcal{T} = T \cup \{\tau_\infty\}$, where τ_∞ represents the outside of the domain, called *dummy element*. Edelsbrunner *et al.* [2] defined the *flow relation* $\prec \subseteq \mathcal{T} \times \mathcal{T}$ with $\tau_1 \prec \tau_2$ if

1. τ_1 and τ_2 share a common triangle ν, and
2. the interior of τ_1 and the circumcenter c of τ_1 lie on the different side of the plane containing ν.

If $\tau_1 \prec \tau_2$, then τ_1 is called the *predecessor* of τ_2; and τ_2 is called the *successor* of τ_1. Here predecessor and successor are only meaningful for a Delaunay tetrahedron. The set of descendants of tetrahedron τ is defined as

$$desc(\tau) = \{\tau\} \cup desc_{\tau \prec \mu}(\mu), \quad \text{where } desc_{\tau \prec \mu}(\tau) = \bigcup_{\tau \prec \mu} desc(\mu).$$

Notice that for a triangle, there is only one successor defined, while there are only at most two successors defined for a tetrahedron. A sequence of tetrahedra with $\tau_1 \prec \tau_2 \ldots \prec \tau_n$ is called a flow path from τ_1 to τ_n, denoted by $\pi(\tau_1, \tau_n)$. See the left figure in Figure 1 for an illustration in 2D.

2.2 Sink Insertion Algorithms

Sink insertion method, proposed by Edelsbrunner *et al.* [2,3], inserts the sinks of bad d-dimensional simplex elements instead of inserting their circumcenters directly. A simplex element is *bad* if its radius-edge ratio is larger than a constant ϱ. For the completeness of the presentation, we review the sink insertion algorithm for a three-dimensional domain.

Algorithm: SINK-INSERTION(ϱ_0)

Empty Encroachment: For any encroached boundary segment, add its midpoint and update the triangulation. For any encroached boundary triangle, add its sink and update the triangulation. If the sink to-be-added encroaches any boundary segment, we split that segment instead of adding that sink.

Bad Elements: For any tetrahedron σ with $\rho(\sigma) > \varrho_0$, find its sink s_σ. Assume that s_σ is the circumcenter of a tetrahedron τ. Insert the sink s_σ to split τ and update the Delaunay triangulation. However, if s_σ encroaches a boundary segment or triangle, we apply the following rules instead of adding s_σ.

Equatorial Sphere: For any boundary triangle μ encroached by the sink s_σ, add the sink s_μ of μ. Update the triangulation accordingly. However, if s_μ encroaches any boundary segment, we apply the following rule instead.

Diametral Sphere: For any boundary segment ν encroached by the sink s_σ or the sink s_μ, add the midpoint of ν. Update the triangulation accordingly.

Recall that the insertion of the circumcenter of a bad d-simplex will immediately remove the simplex. Inserting the sink of a bad tetrahedron may seem counter-intuitive, because the sink of a d-simplex could be far away from it. Consequently, the insertion of the sink may not remove the bad d-simplex immediately. The termination of the algorithm may be in jeopardy. However, it is proved that the circumradius of a tetrahedron σ is no more than that of the tetrahedron τ containing the sink of σ[2,4]. Then the proofs of the termination of Delaunay refinement method [6] can be applied directly to prove the termination of the sink insertion algorithm. If we select $\varrho_0 > 1$, then the minimum distance among mesh vertices after the sink insertion will not decrease, which implies the algorithm's termination. If there are boundary constraints, the constant ϱ_0 has to be increased to $\sqrt{2}$ for $2D$ domain and 2 for $3D$ domain.

3 Good Grading Guarantee in 2D

This section is devoted to study the number of elements in the generated two-dimensional mesh by analyzing the relation between the nearest neighbor function $N()$ defined by the final mesh and the local feature size function *lfs*() defined by the input domain. Here $N(x)$ is the distance from x to its second nearest mesh vertex. A mesh vertex v is always the nearest mesh vertex of itself. Local feature size *lfs*(x) is the radius of the smallest disk centered at x intersecting two non-incident input segments or input vertices. Both $N()$ and *lfs*() are 1-Lipschitz function. A mesh is said to have a *good grading* if the nearest neighbor function $N()$ defined on the mesh is within a constant factor of *lfs*().

We study the spacing relations among intermediate meshes by using similar idea as Ruppert and Shewchuk did. With each vertex v, we associate an *insertion edge length* e_v equal to the length of the shortest edge connected to v immediately after v is introduced into the Delaunay mesh. Here v may not have to be inserted into the mesh actually. For the sake of convenience of analyzing, we also define a *parent vertex* $p(v)$ for each vertex v, unless v is an input vertex. Intuitively, for any non-input vertex v, $p(v)$ is the vertex "responsible" for the insertion of v. We discuss in detail what means by responsible here. If v is inserted as the sink of a triangle σ with $\rho(\sigma) \geq \varrho_0$, then $p(v)$ is the most recently inserted end point of the shortest edge of σ. If all vertices of σ are original input vertices, then $p(v)$ is one of the end points of the shortest edge of σ. If v is the midpoint of an encroached segment, then $p(v)$ is the encroaching vertex. For the sake of simplicity, we always assume that the encroaching vertex is not an input vertex, because Ruppert [5] and Shewchuk [6] showed that the nearest neighbor function $N()$ defined on the Delaunay mesh after enforcing the domain boundary is within

a constant factor of the local feature size function, i.e., $N(v) \sim lfs(v)$. The parent vertex $p(v)$ of v does not need to be inserted into the mesh actually.

We then show that e_v of any introduced mesh vertex v is related to that of its parent vertex $p(v)$. Here v may also not be inserted due to encroaching. For a vertex v, as [6,4], we define the *density ratio* at point v as $D_v = \frac{lfs(v)}{e_v}$. Clearly, D_v is at most one for an input vertex v, and for newly inserted vertex v, D_v tends to become larger. Notice that D_v is defined just immediately after v is introduced to the mesh; it is not defined based on the final mesh.

Lemma 1. [Radius-variation] *Consider a triangle σ. Let u and e_u be the circumcenter and the circumradius of σ. Assume that there is no triangle with a large radius-edge ratio in $desc(\sigma)$ except possibly σ itself. Assume triangle $\tau \neq \sigma$ contains the sink of σ inside. Let μ be the predecessor of triangle τ. Let w and e_w be the circumcenter and circumradius of the triangle μ. Then*

$$e_w - e_u \geq \frac{1}{4\varrho_0}||w - u||.$$

Proof. Let's consider a triangle $\tau_1 = pqr$ and its successor $\tau_2 = pqs$ in $desc(\sigma)$, where $\tau_2 \neq \tau$. Triangle τ_2 has a small radius-edge ratio from the assumption of $desc(\sigma)$. Let x and y be the circumcenter of τ_1 and τ_2 respectively. Let e_x and e_y be the circumradius of τ_1 and τ_2 respectively. The right figure in Figure 1 illustrates the proof that follows. Let c be the midpoint of the edge pq. Let

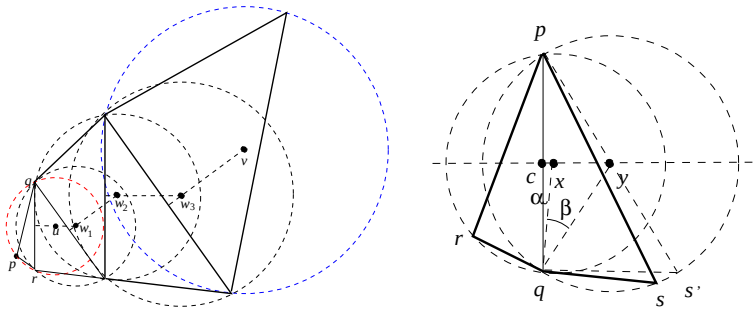


Fig. 1. Left: the descendants of a triangle pqr; Right: the relationship between the circumradius e_x of $\tau_1 = pqr$, the circumradius e_y of $\tau_2 = pqs$ and $||x - y||$.

$\alpha = \angle cqx$ and $\beta = \angle xqy$. Assume that cq has a unit length. Then $e_x = 1/\cos(\alpha)$, $e_y = 1/\cos(\alpha + \beta)$, and $||x - y|| = \tan(\alpha + \beta) - \tan(\alpha)$. It is easy to show that

$$\frac{e_y - e_x}{||x - y||} = \frac{\sin(\alpha + \frac{\beta}{2})}{\cos(\frac{\beta}{2})} = \sin(\alpha + \beta) - \cos(\alpha + \beta)\tan(\frac{\beta}{2}).$$

Assume we fix the value of $\alpha + \beta = \theta_0$, then it is easy to show that

$$\frac{\sin(\alpha + \frac{\beta}{2})}{\cos(\frac{\beta}{2})} \geq \sin(\theta_0) - \cos(\theta_0)\tan(\frac{\theta_0}{2}) = \tan(\frac{\theta_0}{2}).$$

Let ps' be a diameter of the circumcircle of the triangle pqs. It is easy to show that $||q-s|| < ||q-s'|| = 2||c-y|| = 2e_y \sin(\alpha+\beta)$. The triangle $\tau_2 = pqs$ has a small radius-edge ratio implies that $||q-s|| \geq \frac{e_y}{\varrho_0}$. Thus we have $\sin(\alpha+\beta) \geq \frac{1}{2\varrho_0}$. It follows that $\frac{e_y-e_x}{||x-y||} \geq \tan(\frac{\theta_0}{2}) = \frac{\sin\theta_0}{1+\cos\theta_0} > \frac{1}{4\varrho_0}$. Using the triangle inequality, it is easy to show that $e_w - e_u > \frac{1}{4\varrho_0}||w-u||$ by summing up the inequalities for all triangle and successor pairs τ_1 and τ_2 in $desc(\sigma)$ with $\tau_2 \neq \tau$. □

Theorem 1. [Bounded Density Ratio] *There are fixed constants $D_1 \geq 1$ and $D_2 \geq 1$ such that for any vertex v inserted or rejected as the sink of a bad triangle, $D_v \leq D_2$; for any vertex v inserted or rejected as the midpoint of an encroached boundary segment, $D_v \leq D_1$. Hence, there is a constant $D = \max\{D_1, D_2\}$ such that $D_v \leq D$ for all mesh vertex v.*

Proof. We prove this by induction. If v is an original input vertex, then the length e_v of the shortest edge connected to v is at least $lfs(v)$ from the definition of $lfs(v)$. Thus $D_v = \frac{lfs(v)}{e_v} \leq 1$ and the theorem holds.

Then consider non-input vertex v. We first consider inserting v as the sink of a triangle σ. Let u be the circumcenter of the triangle σ. Notice that v is also the sink of any triangle from $desc(\sigma)$. Without loss of generality, assume that no triangle except σ from $desc(\sigma)$ has a large radius-edge ratio. Assume v is the circumcenter of a triangle τ. Notice that e_v is equal to the circumradius of τ.

Case 1: the triangles σ and τ are the same. Notice that σ has a radius-edge ratio at least ϱ_0, then parent p of vertex v is one of the end points of the shortest edge pq of σ. Here p could be the most recently inserted vertex or an original vertex of σ. Then q is an original vertex or is inserted before p. In both cases, we have $e_p \leq ||p-q||$. Then $e_p \leq ||p-q|| \leq \frac{R_\sigma}{\varrho_0}$. And $e_u = R_\sigma \geq \varrho_0 \cdot e_p$. Notice that $lfs(p) \leq D_p e_p$, where D_p is the density ratio bound of vertex p derived from induction. Thus $lfs(u) \leq lfs(p) + ||p-u|| \leq D_p e_p + e_u \leq (\frac{D_p}{\varrho_0} + 1)e_u$. It implies that $D_u = \frac{lfs(u)}{e_u} \leq \frac{D_p}{\varrho_0} + 1$. So a sufficient condition for bounding the density ratio of vertex u is

$$\frac{\max(D_1, D_2)}{\varrho_0} + 1 \leq D_2 \tag{1}$$

Case 2: the triangles σ and τ are not the same. Let w be the circumcenter of the predecessor triangle of τ. Similarly we define e_w as the circumradius of that triangle. Then by previous lemma 1, we know that there is a constant $\delta = \frac{1}{4\varrho_0}$ such that $e_w - e_u \geq \delta||w-u||$. Here w and u could be the same.

Subcase 2.1: vertices w and v are not close, i.e., $||v-w|| \geq \epsilon e_v$, where $\epsilon = \frac{1}{2\varrho_0}$. Then similar to the previous lemma 1, $e_v - e_u \geq \delta||v-u||$. For point v, we have $lfs(v) \leq lfs(u) + ||u-v|| \leq lfs(u) + \frac{1}{\delta}(e_v - e_u) \leq (1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta})e_u + \frac{1}{\delta}e_v$. Thus a sufficient condition that $D_v = \frac{lfs(v)}{e_v} \leq D_2$ is $(1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta})e_u \leq (D_2 - \frac{1}{\delta})e_v$. From $e_u \leq e_v$, this inequality is satisfied if $1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta} \leq D_2 - \frac{1}{\delta}$, and $D_2 - \frac{1}{\delta} \geq 0$. From $D_p \leq \max(D_1, D_2)$, a sufficient condition that $D_v \leq D_2$ is

$$D_2 \geq \max(\frac{1}{\delta}, 1 + \frac{\max(D_1, D_2)}{\varrho_0}). \tag{2}$$

Subcase 2.2: vertices w and v are close, i.e., $||v-w|| < \epsilon e_v$, where $\epsilon = \frac{1}{2\varrho_0}$. For vertex w, similar to subcase 2.1, we have $lfs(w) \leq (1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta})e_u + \frac{1}{\delta}e_w$. Then $lfs(v) \leq lfs(w) + ||v - w||$ implies that $lfs(v) \leq (1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta})e_u + \frac{1}{\delta}e_w + \epsilon e_v \leq (1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta})e_u + (\frac{1}{\delta} + \epsilon)e_v$. Thus, from $e_u \leq e_v$, a sufficient condition that $D_v \leq D_2$ is $1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta} \leq D_2 - \frac{1}{\delta} - \epsilon$, and $D_2 - \frac{1}{\delta} - \epsilon \geq 0$. Consequently, we need

$$D_2 \geq \max(1 + \epsilon + \frac{\max(D_1, D_2)}{\varrho_0}, \frac{1}{\delta} + \epsilon). \tag{3}$$

Case 3: vertex v is the midpoint of a segment that is encroached by a vertex w. Here w is the sink of a triangle σ with large radius-edge ratio. Assume that the sink w is contained in triangle τ. Let u be the circumcenter of triangle σ.

Subcase 3.1: vertices w and u are not same. From the analysis of case 2, we have $lfs(w) \leq (1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta})e_u + (\frac{1}{\delta} + \epsilon)e_w$ by substituting v by w in the results. The vertex w is inside the circumcircle centered at v with radius e_v. Therefore $e_w \leq \sqrt{2}e_v$. From $lfs(v) \leq lfs(w) + ||w - u|| \leq (1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta})e_u + (\frac{\sqrt{2}}{\delta} + \sqrt{2}\epsilon + 1)e_v$, inequality $D_v \leq D_1$ holds if $1 + \frac{D_p}{\varrho_0} - \frac{1}{\delta} \leq D_1 - \sqrt{2}(\frac{1}{\delta} + \epsilon) - 1$ and $D_1 \geq \sqrt{2}(\frac{1}{\delta} + \epsilon) + 1$. Consequently, a sufficient condition is that

$$D_1 \geq \max\{2 + \sqrt{2}(\frac{1}{\delta} + \epsilon) - \frac{1}{\delta} + \frac{\max(D_1, D_2)}{\varrho_0}, \sqrt{2}(\frac{1}{\delta} + \epsilon) + 1\}. \tag{4}$$

Subcase 3.2: vertices u and w are the same. We have $lfs(v) \leq lfs(u) + ||u - v|| \leq (1 + \frac{D_p}{\varrho_0})e_u + e_v \leq (1 + \frac{D_p}{\varrho_0})\sqrt{2}e_v + e_v$. To prove $D_v \leq D_1$, we need

$$\sqrt{2}(1 + \frac{D_p}{\varrho_0}) + 1 \leq D_1. \tag{5}$$

As conclusion, if we choose D_1 and D_2 as the follows,

$$D_1 = \max \begin{cases} 1 + \sqrt{2}(\frac{1}{\delta} + \epsilon), \\ \frac{(\sqrt{2}+1)\varrho_0}{\varrho_0 - \sqrt{2}}, \\ \frac{\varrho_0(2\delta + \sqrt{2} - 1 + \sqrt{2}\epsilon\delta)}{\delta(\varrho_0 - 1)}, \end{cases} \quad \text{and} \quad D_2 = \max(\frac{1}{\delta} + \epsilon, 1 + \epsilon + \frac{D_1}{\varrho_0}),$$

then all inequalities are satisfied. Notice here $\delta = \frac{1}{4\varrho_0}$, and $\epsilon = \frac{1}{2\varrho_0}$. □

The following theorem concludes that the generated mesh has a good grading, i.e., for any mesh vertex v, $N(v)$ is at least some constant factor of $lfs(v)$.

Theorem 2. [Good Grading] *For any mesh vertex v generated by the sink insertion method, the edge incident on v has length at least $\frac{lfs(v)}{D+1}$.*

The proof is omitted here. The values corresponding to D_1 and D_2 guaranteed by the Delaunay refinement method [5,6] are small: $D_v = \frac{lfs(v)}{e_v}$ is at least $\frac{1}{\varrho_0 - \sqrt{2}}$ for a vertex v inserted as the circumcenter of a bad triangle and at least $\frac{\varrho_0}{\varrho_0 - \sqrt{2}}$

for a vertex v inserted as the midpoint of an encroached segment. For instance, Ruppert claims that if the smallest angle is 10 degrees, then no edge is smaller than $\frac{1}{6}$ of the local feature size of either of its endpoints. To guarantee the minimum angle 10 degrees, we need $\varrho_0 = \frac{1}{2\sin 10^o} \approx 2.88$. Then $\delta \approx 0.087$ and $\epsilon \approx 0.174$. So $D_1 \approx 17.54$ and $D_2 \approx 16.69$. It then implies that no edge is smaller than $\frac{1}{19}$ of the local feature size of either of its endpoints in any mesh generated by the sink insertion method. Therefore, the theoretical bound on the number of elements of the mesh generated by sink insertion method is more likely larger than that by the Delaunay refinement method.

Ruppert shows that the nearest neighbor value $N(v)$ of a mesh vertex v of any almost-good mesh is at most a constant factor of $lfs(v)$, where the constant depends on the radius-edge ratio. The above Theorem 2 shows that the nearest neighbor $N(v)$ for the $2D$ Delaunay mesh generated by sink insertion is at least some constant factor of $lfs(v)$. Then we have the following theorem.

Theorem 3. [Linear Size] *The number of triangles in the $2D$ mesh generated by the sink insertion method is within a constant factor of any Delaunay mesh for the same domain, where the constant depends on the radius-edge ratios of the meshes.*

4 Discussions for 3D Domain

Shewchuk [6] showed that the Delaunay refinement method generates almost-good meshes with a good grading guarantee in two and three dimensions. We had showed that the sink insertion method also generates a almost-good mesh with a good grading guarantee in two dimensions. Unfortunately, the proofs presented here can not be directly applied to three dimensions. The reason is as follows. To guarantee the size optimality of the sink insertion method, the nearest neighbor function $N()$ defined on the generated mesh must be within a constant factor of the local feature size function $lfs()$. Notice that $lfs(v) \geq e_v \geq N(v)$, where e_v is the length of the shortest edge connected to v when vertex v is inserted. Therefore, when a vertex v is introduced to the mesh, e_v should be within a constant factor of $lfs(v)$ to guarantee the good grading of the generated mesh. In other words, we have to prove the existence of a constant D such that for each vertex v inserted into the mesh, $lfs(v) \leq De_v$. Or more specifically, there should exist three constants D_1, D_2, and D_3 such that

- for each vertex v inserted as the circumcenter of a tetrahedron, $lfs(v) \leq D_3 e_v$;
- for v inserted as the circumcenter of a boundary triangle, $lfs(v) \leq D_2 e_v$;
- for v inserted as the midpoint of a boundary segment, $lfs(v) \leq D_1 e_v$.

Assume that we insert a vertex v as a sink of a tetrahedron σ, and v is the circumcenter of a tetrahedron τ. Notice that the structure of $desc(\sigma)$ is a DAG whose node out-degree is at most 2. Consider a tetrahedron τ from $desc(\sigma)$. Without loss of generality, we assume that there is no tetrahedron in $\pi(\sigma, \tau)$ with a large radius-edge ratio except tetrahedron σ itself. In three dimensions,

the fact that a tetrahedron has a small radius-edge ratio does not guarantee that the tetrahedron has no small angles. Slivers are the only tetrahedra that have a small radius-edge ratio but their aspect ratio could be arbitrarily large. Let u and e_u be the circumcenter and the circumradius of σ. Let e_v be the circumradius of the tetrahedron τ. It is possible that e_v is almost the same as e_u even vertex v is far away from u. Consequently, $\frac{e_v - e_u}{||v-u||}$ could not be bounded from below by any constant. Figure 2 gives an example of a configuration such that $\frac{e_v - e_u}{||v-u||}$ could be arbitrarily small. This, together with the fact that $lfs(v)$ could be as large as $lfs(u) + ||v-u||$ implies that $D_v = \frac{lfs(v)}{e_v}$ could be much larger than $D_u = \frac{lfs(u)}{e_u}$. Therefore we can not bound the density D_v using the relation between D_u and D_v even assuming that the density D_u of vertex u is bounded by a constant D_3.

Figure 2 is constructed as follows. Let sliver $p_0q_0r_0s_0$ be a successor of the tetrahedron σ, which is not shown in the left figure of Figure 2. Assume that the circumcenter of the sliver $p_0q_0r_0s_0$ is on the different side of the plane $H_{p_0r_0s_0}$ passing p_0, r_0 and s_0 with sliver $p_0q_0r_0s_0$. One of the successor of sliver $p_0q_0r_0s_0$ is constructed by lifting vertex q_0 to a new position q_1 such that tetrahedron $p_0q_1r_0s_0$ is a sliver and its circumcenter is on the different side of the plane $H_{p_0q_1s_0}$ passing p_0, q_1 and s_0 with the tetrahedron $p_0q_1r_0s_0$. Then we lift the node r_0 to r_1 to construct a sliver successor $p_0q_1r_1s_0$ of tetrahedron $p_0q_1r_0s_0$ whose circumcenter is on the different side of the plane $H_{p_0q_1r_1}$ with $p_0q_1r_1s_0$. We then construct a sliver successor $p_0q_1r_1s_1$ of $p_0q_1r_1s_0$ by lifting s_0 to s_1. Sliver successor $p_1q_1r_1s_1$ is constructed by lifting p_0 to a new position p_1. The above procedure could be repeated many rounds if $p_0q_0r_0s_0$ is carefully configured and every lifting is carefully chosen. The middle figure in Figure 2 give the sliver pattern used in constructing this example. It is easy to show that using only the tetrahedron τ also can not bound the density ratio $D_v = \frac{lfs(v)}{e_v}$. Assume that p is a vertex of the shortest edge of tetrahedron τ; see the right figure in Figure 2. Then we have $D_v \leq 1 + \frac{4}{\sqrt{6}} D_p$. Here the situation $D_v = 1 + \frac{4}{\sqrt{6}} D_p$ could be

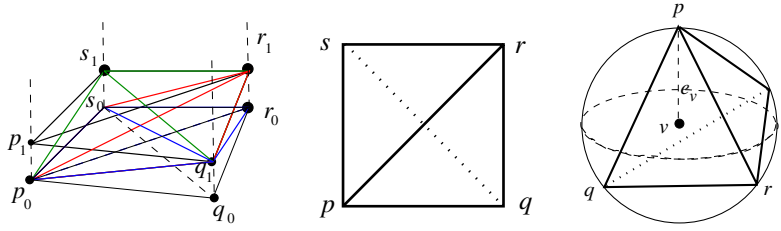


Fig. 2. Left: all sliver descendants; Right: the sink simplex does not help.

achieved if τ is a regular tetrahedron. It implies that the upper bound for D_v could be always larger than that for D_p. However, we doubt that these situations can really happen in practice.

5 Conclusion

In this paper, we show that the sink insertion method guarantees to generate a two-dimensional mesh with good grading. On the other hand, we also give an example of three-dimensional local mesh configuration to show that the sink insertion method may fail to generate a mesh with size $O(n)$, where n is the minimum number of the mesh elements with the same radius-edge ratio quality.

As reported by the experimental results (Guoy and Edelsbrunner, private communication), the sink insertion method usually generates meshes whose sizes are not much larger than that by Delaunay refinement method. However, it is interesting to see if we can theoretically prove the good grading guarantee of the sink insertion method or give an example of three-dimensional domain such that a sequence of sink insertions will generate a mesh whose size is larger than $O(n)$. Notice Li [4] recently gave a new refinement-based algorithm that generates well-shaped three-dimensional meshes with size $O(n)$. Li [4] also proposed a variation of the sink insertion method, which inserts a point near the sink and its insertion will not introduce small slivers compared to τ instead of inserting the sink of a tetrahedron τ with large radius-edge ratio or sliver. This variation guarantees to generate well-shaped 3D Delaunay meshes. However, it is open whether this variation will have a good grading guarantee. It is interesting to see what is the mesh size relation between two meshes generated by the sink insertion method and this variation proposed in [4].

References

1. Chew, L. P. Constrained Delaunay triangulations. *Algorithmica 4* (1989), 97–108.
2. Edelsbrunner, H., Facello, M. A., and Liang, J. On the definition and the construction of pockets in macromolecules. *Diecrete Applied Mathematics 88* (1998), 83–102.
3. Edelsbrunner, H., and Guoy, D. Sink-insertion for mesh improvement. *Symposium of Computational Geometry* 2001.
4. Li, X. Y. *Sliver-free Three Dimensional Delaunay Mesh Generation.* PhD thesis, University of Illinois at Urbana-Champaign, 2000.
5. Ruppert, J. A new and simple algorithm for quality 2-dimensional mesh generation. In *Third Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 83–92.
6. Shewchuk, J. R. Tetrahedral mesh generation by Delaunay refinement. In *14th Annual ACM Symposium on Computational Geometry* (1998), pp. 86–95.
7. Talmor, D. *Well-Spaced Points for Numerical Methods.* PhD thesis, Carnegie Mellon University, 1997.

Polynomial Time Algorithms for Three-Label Point Labeling*

Rob Duncan[1], Jianbo Qian[2], and Binhai Zhu[1]

[1] Department of Computer Science,
Montana State University, Bozeman, MT 59717-3880, USA.
{duncan,bhz}@cs.montana.edu.
[2] Department of Mathematics,
Shandong University, Jinan, China.
qian@math.sdu.edu.cn.

Abstract. In this paper, we present an $O(n^3 \log n)$ time solution for the following multi-label map labeling problem: Given a set S of n distinct sites in the plane, place at each site a triple of uniform squares of maximum possible size such that all the squares are axis-parallel and a site is on the boundaries of its three labeling squares. We also study the problem under the discrete model, i.e., a site must be at the corners of its three label squares. We obtain an optimal $\Theta(n \log n)$ time algorithm for the latter problem.

1 Introduction

Map labeling is a popular problem on information visualization in our daily life. It is an old art in cartography and finds new applications in recent years in GIS, graphics and graph drawing [AKS98, CMS93, CMS95, DF92, FW91, Imh75, Jon89],
[KR92, KT97, KT98a, PZC98, Wa94, WW95, ZP99]. Among many problems in map labeling, labeling points is of special interest to many practitioners and theoreticians. In the paper by Formann and Wagner, any point site can only be labeled with 4 candidate (axis-parallel) squares each of which has a vertex anchored at the site and the objective is to maximize the square size. See Figure 1, I.a, for an instance of the problem. (In general we call this kind of model *discrete*, i.e., each site has $O(1)$ candidates.) Even this seemingly simplest version is shown to be NP-complete and moreover; it is NP-hard to approximate within factor 2 [FW91]. (For details regarding NP-completeness readers are referred to [GJ79].) In the past several years more generalized models have been proposed. The basic idea is to allow each site to have an infinite number of possible candidate labels (see [DMMMZ97, IL97, KSW99, SW99, ZP99]). This model is more natural than the previous discrete models (like the one in [FW91]) and has been

* This research is supported by Research Grants Council of Hong Kong SAR, China (CERG Project No. CityU1103/99E). Part of this research was done when the last two authors were with City University of Hong Kong.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 191–200, 2001.
© Springer-Verlag Berlin Heidelberg 2001

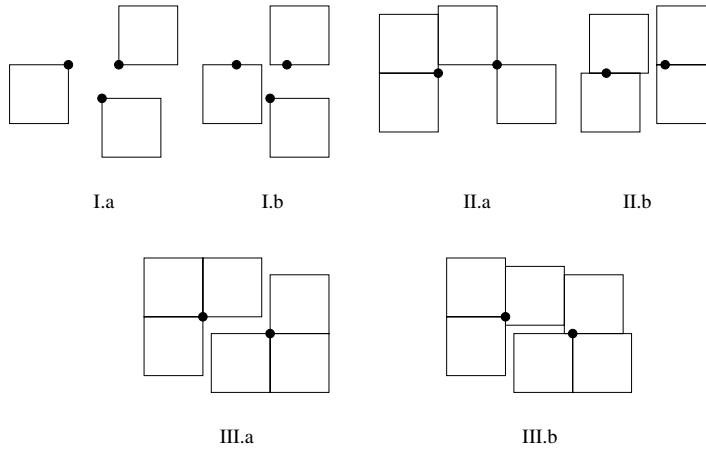


Fig. 1. Examples for one-label, two-label and three-label point labeling.

coined as the *sliding model* in [KSW99]. On the other hand, designing efficient algorithms for map labeling under the sliding model is a new challenge to map labeling researchers. We briefly review some recent results on labeling points under the sliding model.

Before our review, we briefly define some necessary concepts in approximation algorithms. An approximation algorithm for a (maximization) optimization problem Π provides a **performance guarantee** of ρ if for every instance I of Π, the solution value returned by the approximation algorithm is at least $1/\rho$ of the optimal value for I. For the simplicity of description, we simply say that this is a factor ρ approximation algorithm for Π.

In [DMMMZ97], Doddi et al. designed several approximation algorithms for labeling points with arbitrarily oriented squares and circles (though the constant factors are impractical: 36.6 and 29.86 respectively). They also presented bicriteria PTAS for the problems. The former bound was improved to 12.95 by Zhu and Qin [ZQ00] and significantly to 5.09 by Doddi et al. most recently [DMM00]. The latter bound on labeling points with circles was improved to 19.35 by Strijk and Wolff [SW99] and recently to 3.6 by Doddi et al. [DMM00]. If any labeling square must be along a fixed direction (e.g., axis-parallel), Zhu and Qin showed that it is possible to have a factor-4 approximation [ZQ00]. In [KSW99] van Kreveld et al. proved that it is NP-hard to decide whether a set of points can all be labeled with axis-parallel unit squares under the sliding model. See Figure 1, I.b, for an example of labeling points with sliding axis-parallel squares. (In fact, in [KSW99], van Kreveld et al. tried to maximize the number of sites labeled instead of the size of the labels.) In [SW99] Strijk and Wolff used similar ideas to prove that the problem of labeling points with maximum size uniform circles is NP-hard. This explains why it is meaningful to study approximation algorithms for these problems.

As another kind of generalization for map labeling, recently Zhu and Poon studied the problem of labeling point sites with axis-parallel uniform square pairs and circle pairs. The motivation is that in some applications we need two labels for a site [KT98] (like labeling a map for weather reporting or labeling a bilingual city map). They obtained factor-4 and factor-2 algorithms for the two problems respectively, besides presenting a bicriteria approximation scheme [ZP99]. For map labeling with uniform square pairs, Zhu and Qin [ZQ00] first improved the approximation factor of [ZP99] from 4 to 3. Then Qin et al. further improved the factor to 2 [QWXZ00]. More recently, Spriggs proved that the problem is NP-hard and, in fact, NP-hard to approximate within a factor of 1.33 [Sp00]. See Figure 1, II.a and II.b for examples of labeling points with uniform square pairs under the discrete and sliding model respectively.

For map labeling with uniform circle pairs, Qin et al. first improved the 2 approximation factor to 1.96 and proved that problem is NP-hard, in fact, NP-hard to approximate within a constant factor of $\delta > 1$ [QWXZ00]. The 1.96 factor was recently improved to 1.686 by Spriggs and Keil [SK00] and then by Wolff et al. to 1.5 [WTX00]. There are still some gaps between the lower and upper bounds for both of the two problems.

In this paper, we study the problem of labeling point sites with uniform square triples. (See Figure 1, III.a, III.b for examples of labeling points with uniform square triples under the discrete and sliding model respectively.) The problem is interesting both in application and theory. In practice, many weather reporting programs on TV need to label a city with three labels: its name, temperature and chance of rainfall. In theory, the problem of labeling point sites with uniform squares is NP-hard under either the discrete model [FW91] or the sliding model [KSW99]. Also, the problem of labeling point sites with uniform square pairs is NP-hard under both the discrete and sliding model [Sp00]. On the other hand, labeling point sites with four squares is trivially polynomial solvable (the solution is decided by the closest pair of the point set, under the L_∞ metric). The labeling problems for points with circles and circle pairs are both NP-hard [SW99, QWXZ00]. Finally, we remark that it is impossible to label a point with three or more non-overlapping circles.

We now define formally the problems to be studied. We also make some extra definitions related to our algorithms. The MLUST problem (Map Labeling with Uniform Square Triples) is defined as follows:

Instance: A set S of points (sites) p_1, p_2, ..., p_n in the plane.
Problem: Does there exist a set of n triples of axis-parallel squares of maximum size (i.e., length of a side) L^* each of which are placed at each input site $p_i \in S$ such that no two squares intersect and no site is contained in any square.

Notice that we can have two versions of the problem: the problem can be under the discrete model in which every site must be at the corners of its three labeling squares and the problem can also be under the sliding model in which every site can be on the boundaries of its three labeling squares. Because of the

nature of the problem even under the sliding model every site must be at the corners of at least two of its labeling squares. We present below a few extra definitions which will be used in later sections.

Given a set S of n sites in the plane, the closest pair of S under L_∞ metric, $D_\infty(S)$, is defined as the minimum L_∞-distance between any two points in S. Clearly $D_\infty(S)$ can be computed in $O(n \log n)$ time with standard algorithm in computational geometry [PS85].

2 MLUST under the Discrete Model

In this section we present the details of a polynomial time algorithm for the MLUST problem under the discrete model. Because of the nature of the problem, the metric discussed in this paper is L_∞ unless otherwise specified. Let $D_\infty(S)$ be the closest pair of S under the L_∞-metric. Let l^* denote the size of each square in the optimal solution of the discrete MLUST problem. The following lemma is easy to prove.

Lemma 1. $D_\infty(S)/2 \leq l^* \leq D_\infty(S)$.

In the following we show how to decide whether a set of points S can be labeled with square triples with edge length l, $D_\infty(S)/2 \leq l \leq D_\infty(S)$. For any two points $p_i, p_j \in S$, let $d_\infty(p_i, p_j)$ denote the L_∞-distance between them. Let C_i denote the L_∞-circle centered at point $p_i \in S$ with radius l. Clearly, the circle C_i contains no other point from the input set S except its own center. Note that each circle C_i can be partitioned into four L_∞-circles with radius $l/2$, which are geometrically squares with edge length l. We loosely call them *sub-squares* of C_i. Basically, to label p_i we need to select three out of four sub-squares in each C_i so that no two sub-squares intersect each other.

We now present our algorithm. First, we compute the following multiple intersection graph $G_M(S, l)$. C_i $(1 \leq i \leq n)$ are the vertices for $G_M(S, l)$. There is an edge between C_i, C_j if they intersect and only one pair of sub-squares of them overlap. If at least two (and at most three) pairs sub-squares of C_i, C_j overlap, then we draw a multiple of two edges between C_i, C_j. (Recall that two squares overlap if they have a common interior point.)

Lemma 2. *There is a valid labeling for S with square triples of size l if and only if every connected component of $G_M(S, l)$ has at most one cycle.*

Proof: We refer to Figure 2. By the definition of the problem, if we have a cycle in $G_M(S, l)$ which contains no multiple edge then when we label p_i the labeling will generate 'pressure' to either C_j or C_k. In other words, one of the sub-squares of either C_j or C_k will be 'destroyed' and cannot be used as legal label for either p_j or p_k anymore. This holds for all the nodes involved in that cycle. (In Figure 2, Case 1, the labeling of p_i generates 'pressure' on C_j.) If C_i, C_j form a cycle with two multiple edges, similar claim holds, except that we need to label p_i (p_j) with a sub-square which only generates one 'pressure' to C_j (C_i). (See Figure 2, Case 2.) Now we continue with our proof.

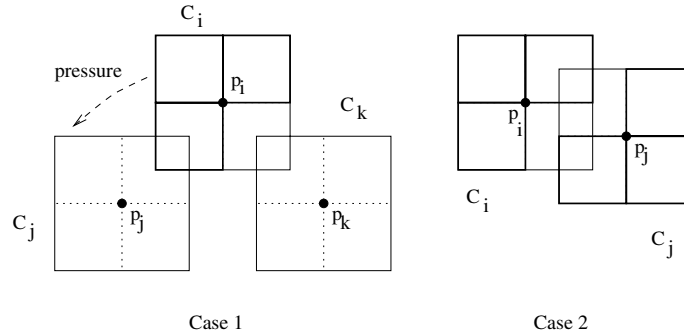


Fig. 2. Illustration for the proof of Lemma 2.

(*Necessity.*) Assume that there is a valid labeling for S with square triples of size l. Then for each C_i, it receives at most one 'pressure', i.e., at most one of its four sub-squares cannot be used as legal label for p_i. Clearly, that implies that every connected component of $G_M(S, l)$ contains at most one cycle.

(*Sufficiency.*) It is easy to see that a leaf dangling at a cycle has no influence over the labeling of the centers of those nodes involved in a cycle. So we assume that there is no leaf node in any connected component of $G_M(S, l)$. Now if each connected component of $G_M(S, l)$ is a cycle then obviously we can label the centers of all those nodes C_i involved in that cycle. □

Clearly the graph $G_M(S, l)$ has a vertex degree of at most 12. So the graph is of linear size and checking whether the graph contains more than one cycle can be done in linear time with standard graph algorithm. To obtain a polynomial time solution for MLUST under the discrete model all we need to do is to prove that there are only a polynomial number of candidates for l. Among them, the largest will give us the size of the optimal solution. Let the coordinates of p_i be $(x(p_i), y(p_i))$ and let $d_{min}(p_i, p_j)$=min$\{|x(p_i)-x(p_j)|, |y(p_i)-y(p_j)|\}$. (Note that $d_\infty(p_i, p_j)$=max$\{|x(p_i) - x(p_j)|, |y(p_i) - y(p_j)|\}$.) We have the following lemma.

Lemma 3. *The size of the optimal solution for MLUST under the discrete model is equal to either $d_\infty(p_i, p_j)$ (in which case it is exactly $D_\infty(S)$), or $d_\infty(p_i, p_j)/2$ or $d_{min}(p_i, p_j)$; provided that its value is bounded by $D_\infty(S)/2$ and $D_\infty(S)$.*

Lemmas 1, 2 and 3 naturally give us the following algorithm. For each site p_i, we look at the axis-parallel square $\mathcal{C}$ centered at p_i with edge length $4D_\infty(S)$. Clearly any point p_k out of this square would be at a distance longer than $2D_\infty(S)$, which implies $d_\infty(p_i, p_k)/2$ cannot be the optimal solution value for the problem. As any two sites in $\mathcal{C}$ must be at least $D_\infty(S)$ distance away, we only need to consider a constant number (24) points in S which are the closest to p_i. (Overall, for all p_i this can be computed in $O(n \log n)$ time using standard techniques [DLSS93].) For each such point p_j, we simply measure $d_\infty(p_i, p_j)$ and $d_{min}(p_i, p_j)$. If $d_\infty(p_i, p_j)/2$ or $d_{min}(p_i, p_j)$ is out of the range $[D_\infty(S)/2, D_\infty(S)]$

then throw it away as a valid candidate. Eventually we have at most $2 \times 24n + 1 = O(n)$ number of candidates. We sort them into a list $l_1, ..., l_{O(n)}$ and then we run a binary search over this list to decide the maximum value l^* such that a valid labeling for S with such a value, exists. As the decision step, following Lemma 2, takes $O(n)$ time, the whole algorithm takes $O(n \log n)$ time. It is easy to show that $\Omega(n \log n)$ is the lower bound for the MLUST problem under either the discrete or the sliding model, by a reduction from the element uniqueness problem. Summarizing the above results, we have the following theorem.

Theorem 1. *For any given set of n points in the plane, the above algorithm, which runs in $\Theta(n \log n)$ time, produces an optimal solution for the MLUST problem under the discrete model.*

3 MLUST under the Sliding Model

In this section, we shall proceed with the more interesting problem of labeling a set of points with sliding square triples. Because of the nature of the problem, not all of the three labels for a site p_i could slide; in fact, only one of them could. We hence call the two discrete sub-squares in the label of p_i *base sub-squares* or *base labels*. Let L^* be the optimal solution of the problem MLUST under the sliding model. We have a lemma similar to Lemma 1.

Lemma 4. $D_\infty(S)/2 \leq L^* \leq D_\infty(S)$.

Our general idea is the same as that for the discrete case. We first try to design a decision procedure which can decide for any $l \in [D_\infty(S)/2, D_\infty(S)]$ whether a valid labeling of S with sliding square triples of size l exists. We follow the same procedure in the previous section to build a multi-graph $G_M(S, l)$. (The nodes of $G_M(S, l)$ are the set of C_i whose centers are p_i, for all $p_i \in S$.) However, because of the difference between the two models we cannot immediately have a lemma similar to Lemma 2. The reason is that the sliding of some label for p_i might simply terminate any 'pressure' its neighbor carries over to it. In Figure 3 (a), if we label p_i using the shaded sliding label then the 'pressure' from p_k vanishes and

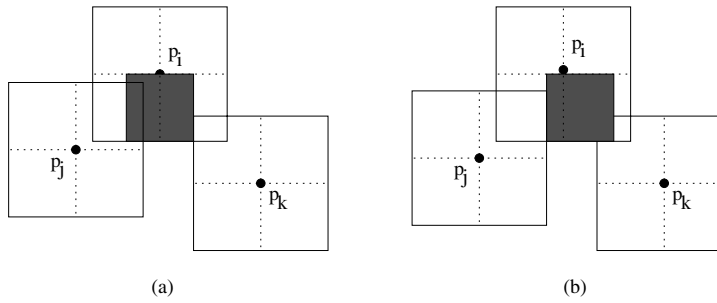


Fig. 3. A pressure-releasing operation.

in Figure 3 (b), if we label p_i using the shaded sliding label then the 'pressure' from p_j vanishes. We call p_j, p_i, p_k a critical triple if C_j intersects C_i and C_i intersects C_k. A *pressure-releasing* operation on a critical triple p_j, p_i, p_k is that we label p_i with sliding labels such that the labels generate minimum 'pressure' on either p_j or p_k, i.e., the number of either p_j or p_k's sub-squares destroyed by the labels of p_i is minimized and furthermore, the area of both p_j and p_k's sub-squares destroyed by the labels of p_i is also minimized. (The first condition implies that a pressure-releasing operation destroys either 0 or 1 sub-square of p_j or p_k. The second condition implies that if a pressure-releasing operation has to destroy a sub-square of p_j or p_k then it will destroy the minimum area of it. Finally, it is clear that we can perform $O(1)$ number of pressure-releasing operations on any given critical triple.) In this case we call C_i a *cycle-breaker* in $G_M(S, l)$ and clearly a cycle-breaker will terminate some pressure along some cycle in $G_M(S, l)$ if we perform a pressure-releasing operation on its center. Therefore, we have the following revised version of Lemma 2, whose proof is straightforward.

Lemma 5. *There is a valid labeling for S with sliding square triples of size l if and only if every connected component of $G_M(S, l)$ has at most one cycle after a set of pressure-releasing operations are enumerated.*

Notice that different from the discrete MLUST problem, this lemma does not give us a static algorithm as does Lemma 2. In fact, at the first glance, it seems that the above lemma gives us an exponential solution. However, we will make use of the specific property of $G_M(S, l)$ to obtain a polynomial time solution.

What we do is as follows. As the vertex degree of any node in $G_M(S, l)$ is at most a constant (12), we can fix any node C_i and identify all possible states of it. That state is determined by the position of the two base sub-squares as well as along which path adjacent to C_i the 'pressure' will be generated. Clearly we have $O(1)$ number of states for C_i.

What we do next is to fix a state of C_i and traverse the graph by following those paths carrying 'pressures' generated so far. Suppose that after visiting C_j and successfully labeling p_j (i.e., the current state of p_j is *safe*), we reach at a vertex C_k. If we can find a valid labeling of p_k taking into consideration all the pressures generated on p_k so far, then we set the state of p_k as safe and we continue our traversal. If we reach a *dead* state at p_k, i.e., no valid labeling of p_k exists (in other words, at least two of p_k's sub-squares are destroyed), then we backtrack to C_j and traverse the edge (C_j, C_k) by starting at a different safe state of p_j, if there exists one. If we backtrack to p_i and try out all its safe states and still cannot find a valid labeling for all the sites in S, then a valid labeling of S with square triples of size l does not exist.

At the first sight, this procedure seems to take exponential time. However, the following lemma guarantees a polynomial time solution for fixed l. (It is obvious that if we can label all sites corresponding to nodes in $G_M(S, l)$ with square triples of size l then there must exist at least one node in $G_M(S, l)$ which admits a pressure-releasing operation.)

Lemma 6. *In the above procedure, if at each cycle-breaker C_j we minimize the out-going pressure along (C_j, C_k) while withholding the incoming 'pressure' then for each state of C_j there is a unique state for the next cycle-breaker C_k in the same cycle.*

The above lemma basically shows that the procedure in the previous paragraph runs in linear time if we start at a C_i which admits a pressure-releasing operation. (For each cycle-breaker C_j we have $O(1)$ state, when traversing the graph from C_j to the next cycle-breaker C_k we can only reach exactly one state of C_k. This procedure finishes either after we try all states of C_i without finding a valid labeling for S or we terminate with a valid labeling of size l for set S.) However, as in the optimal labeling not all C_i's admit a pressure-releasing operation we need to try the above procedure $O(n)$ times — starting at every possible node in the graph. Therefore, for a fixed l deciding whether we can label S with sliding square triples of size l can be done in $O(n \times n) = O(n^2)$ time. To solve MLUST under the sliding model, we must also make sure that there are only a polynomial number of candidates for L^*. This is guaranteed with the following lemma.

Lemma 7. *The size of the optimal solution for MLUST under the sliding model L^* is equal to either the optimal solution for MLUST under the discrete model or $d_\infty(p_i, p_j)/K$ for some $p_i, p_j \in S$ and some K such that $1 \leq K \leq n-2$, provided that its value is bounded by $D_\infty(S)/2$ and $D_\infty(S)$.*

Proof. It is only necessary to discuss the situation when L^* is not equal to the optimal solution for MLUST under the discrete model. In this situation, the reason that the optimal solution value L^* for MLUST under the sliding model cannot be increased is that there exists a series of K labels $p'_1, ..., p'_K$ touching each other and the sum of their sizes is exactly the distance between some sites p_i, p_j, i.e., each p'_k contributes some distance to fill the L_∞-gap between p_i, p_j (Figure 4). We call p_i, p_j an extreme pair. (Note that in Figure 5 the base labels for all the sites are not shown.) What remains to show is that each p'_k contributes exactly a distance of L^* to fill the L_∞-gap between p_i, p_j, which in turn implies that we do not need to consider any K which is larger than $n-2$. Assume to the contrary that this is not the case, i.e., at least one of the sites, say p'_k, would contribute $2L^*$ to fill the L_∞-gap between p_i, p_j. However, this implies that either p_i, p'_k or p'_k, p_j would be an extreme pair. □

Similar to the previous section, Lemmas 4, 5, 6 and 7 naturally give us the following algorithm. For each pair of sites p_i, p_j, we simply measure $d_\infty(p_i, p_j)$. If any of $d_\infty(p_i, p_j)/K (1 \leq K \leq n-2)$ is out of the range $[D_\infty(S)/2, D_\infty(S)]$ then we throw it away as a valid candidate. With i, j fixed, we have at most $O(n)$ number of candidates. In total we have $O(n^3)$ number of candidates for L^*. (We also need to test all the $O(n)$ candidates for the discrete problem.) We sort them into a list $L_1, ..., L_{O(n^3)}$ in $O(n^3 \log n)$ time and then we run a binary search over this list to decide the maximum value L^* such that a valid labeling for S with such a value, exists. As the decision step, following Lemma 6, takes

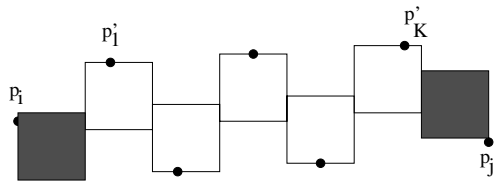


Fig. 4. Illustration for the proof of Lemma 7.

$O(n^2)$ time, the whole algorithm takes $O(n^3 \log n + n^2 \times \log n^3) = O(n^3 \log n)$ time. Summarizing the above results, we have the following theorem.

Theorem 2. *For any given set of n points in the plane, there is an $O(n^3 \log n)$ time solution for the MLUST problem under the sliding model.*

4 Concluding Remarks

In this paper, we present polynomial time algorithms for the problem of labeling point sites with uniform square triples, under either the discrete or sliding models. This is significantly different from the problem of labeling point sites with uniform square pairs, which is NP-hard under both the discrete and sliding models. An immediate question is whether we can reduce the gap between the $\Omega(n \log n)$ lower bound and the $O(n^3 \log n)$ upper bound for the general problem (under the sliding model).

References

[AKS98] P. Agarwal, M. van Kreveld and S. Suri. Label placement by maximum independent set in rectangles. *Comp. Geom. Theory and Appl.,* 11:209-218, 1998.

[CMS93] J. Christensen, J. Marks, and S. Shieber. Algorithms for cartographic label placement. In *Proc. 1993 ASPRS/ACSM Annual Convention and Exposition*, volume 1, pages 75–89, 1993.

[CMS95] J. Christenson, J. Marks, and S. Shieber. An Empirical Study of Algorithms for Point-Feature Label Placement, *ACM Transactions on Graphics,*14:203-222, 1995.

[DF92] J. Doerschler and H. Freeman. A rule-based system for cartographic name placement. *CACM*, 35:68–79, 1992.

[DLSS93] A. Datta, II.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems, In *Proc. 3rd Worksh. Algorithms and Data Structures,* springer-verlag, LNCS 709, pages 265-276, 1993.

[DMM00] S. Doddi, M. Marathe and B. Moret, Point set labeling with specified positions. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 182-190, June, 2000.

[DMMMZ97] S. Doddi, M. Marathe, A. Mirzaian, B. Moret and B. Zhu, Map labeling and its generalizations. In *Proc. 8th ACM-SIAM Symp on Discrete Algorithms (SODA'97)*, New Orleans, LA, pages 148-157, Jan, 1997.

[FW91] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 281–288, 1991.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, CA, 1979.

[Imh75] E. Imhof. Positioning names on maps. *The American Cartographer*, 2:128–144, 1975.

[IL97] C. Iturriaga and A. Lubiw. Elastic labels: the two-axis case. In *Proc. Graph Drawing'97*, pages 181–192, 1997.

[Jon89] C. Jones. Cartographic name placement with Prolog. *Proc. IEEE Computer Graphics and Applications*, 5:36–47, 1989.

[KR92] D. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM J. Disc. Math.*, 5:422–427, 1992.

[KSW99] M. van Kreveld, T. Strijk and A. Wolff. Point set labeling with sliding labels. *Comp. Geom. Theory and Appl.,* 13:21-47, 1999.

[KT97] K. Kakoulis and I. Tollis. An algorithm for labeling edges of hierarchical drawings. In *Proc. Graph Drawing'97*, pages 169–180, 1997.

[KT98a] K. Kakoulis and I. Tollis. A unified approach to labeling graphical features. *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 347–356, 1998.

[KT98] K. Kakoulis and I. Tollis. On the multiple label placement problem. In *Proc. 10th Canadian Conf. on Comput. Geom.*, pages 66–67, 1998.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[PZC98] C.K. Poon, B. Zhu and F. Chin, A polynomial time solution for labeling a rectilinear map. *Inform. Process. Lett.*, 65(4):201-207, Feb, 1998.

[QWXZ00] Z.P. Qin, A. Wolff, Y. Xu and B. Zhu, New algorithms for two-label point labeling. *Proc. 8th European Symp. on Algorithms (ESA'00)*, pages 368–379, Springer-Verlag, LNCS series 1879, 2000.

[Sp00] M. Spriggs. On the complexity of labeling maps with square pairs. *manuscript*, 2000.

[SK00] M. Spriggs and M. Keil. A new bound for map labeling with uniform circle pairs. *Inform. Process. Lett.*, submitted, 2000.

[SW99] T. Strijk and A. Wolff. Labeling points with circles. Tech Report B 99-08, Institut fur Informatik, Freie Universitat, Berlin, April, 1999.

[Wa94] F. Wagner. Approximate map labeling is in $\Omega(n \log n)$. *Inform. Process. Lett.*, 52:161–165, 1994.

[WTX00] A. Wolff, M. Thon and Y. Xu. "A better lower bound for two-circle point labeling," In *Proc. 11th Intl Symp. on Algorithms and Computation (ISAAC'00),* pages 422–431, Springer-Verlag, LNCS series 1969, 2000.

[WW95] F. Wagner and A. Wolff. Map labeling heuristics: Provably good and practically useful. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 109–118, 1995.

[ZP99] B. Zhu and C.K. Poon. "Efficient approximation algorithms for multi-label map labeling," In *Proc. 10th Intl Symp. on Algorithms and Computation (ISAAC'99),* pages 143–152, Springer-Verlag, LNCS series 1741, 1999.

[ZQ00] B. Zhu and Z.P. Qin. New approximation algorithms for map labeling with sliding labels. *J. of Combinatorial Optimization*, accepted, 2000.

Approximation Algorithms for the Watchman Route and Zookeeper's Problems

Xuehou Tan

Tokai University, 317 Nishino, Numazu 410-0395, Japan
tan@fc.u-tokai.ac.jp

1 Preliminary

Given a simple polygon P with n vertices and a starting point s on its boundary, the *watchman route problem* asks for a shortest route in P through s such that each point in the interior of the polygon can be seen from at least one point along the route. It is known that the watchman route problem can be reduced in $O(n \log n)$ time to that of computing the shortest route which visits a set of line segments in polygon P. In this paper, we present a simple approximation algorithm for computing the shortest route visiting that set of line segments. Our algorithm runs in $O(n)$ time and produces a watchman route of at most 2 times the length of the shortest watchman route. The best known algorithm for computing the shortest watchman through s takes $O(n^4)$ time [3]. Our scheme is also employed to give a $\sqrt{2}$-approximation solution to the *zookeeper's problem*, which is a variant of the watchman route problem.

The known watchman route and zookeeper's algorithms primarily make use of the *reflection principle*. If a and b are two points on the same side of a line L, then the shortest path visiting a, L and b in that order, denoted by $S(a, L, b)$, follows the reflection principle, i.e., the incoming angle of path $S(a, L, b)$ with L is equal to the outgoing angle of $S(a, L, b)$ with L. To actually compute the path $S(a, L, b)$, we first reflect b across L to get its image b', then draw the straight segment $\overline{ab'}$ from a to b', and finally fold back the portion of $\overline{ab'}$ lying in the other side of L to obtain the path $S(a, L, b)$. See Fig. 1a for an example. More generally, to find a shortest path from a to b that visits a segment l, denoted by $S(a, l, b)$, we first compute the shortest path that passes between the endpoints of segment l and connects a to the reflection b' and then fold back the portion of the path in the other side of l (Fig. 1b).

Let $L(a)$ denote the point of L nearest to a (Fig. 1). The path consisting of the segments $\overline{aL(a)}$ and $\overline{L(a)b}$, denoted by $S'(a, L, b)$, is a good approximation of $S(a, L, b)$. Note that a, $L(a)$ and b' form an obtuse-angled triangle. Since the sum of lengths of two shorter edges of a right-angled triangle is the maximum when two acute angles are $\pi/4$, we have the following result.

Lemma 1 *For an obtuse-angled triangle, the sum of lengthes of two shorter edges is smaller than or equal to $\sqrt{2}$ times the length of the longest edge.*

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 201–206, 2001.
© Springer-Verlag Berlin Heidelberg 2001

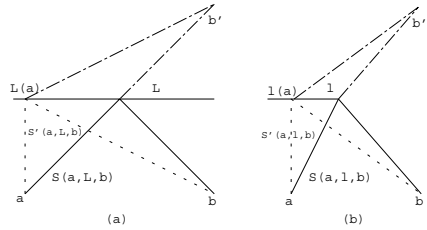


Fig. 1. The reflection principle and its approximation.

Analogously, let $l(a)$ denote the point of the segment l nearest to a (Fig. 1b). Then the path consisting of two segments $\overline{al(a)}$ and $\overline{l(a)b}$, denoted by $S'(a, l, b)$, is also a $\sqrt{2}$-approximation of the path $S(a, l, b)$.

2 Approximating the Shortest Watchman Route

Let P be a polygon with a point s on its boundary. A vertex is *reflex* if its internal angle is greater than π. Polygon P can be partitioned into two pieces by a "cut" that starts at a reflex vertex v and extends either edge incident to v until it first hits the boundary. We say a cut is a *visibility cut* if it produces a *convex* angle ($< \pi$) at v in the piece of P containing s. For a cut C, the piece of P containing s, denoted by $P(C)$, is called the *essential piece* of C. We say cut C_j *dominates* cut C_i if $P(C_j)$ contains $P(C_i)$. A cut is called an *essential cut* if it is not dominated by any other cuts. Clearly, a route is a watchman route if and only if it visits all essential cuts. (All essential cuts can be computed in $O(n \log n)$ time.)

Let C_1, C_2, $\cdots$, C_m the sequence of essential cuts indexed in the clockwise order of their left endpoints, starting at s. Also, let $s = s_0 = s_{m+1}$. First, we find the point of C_1 that is closest to s_0 in $P(C_1)$. Let s_1 denote the point found on C_1, which is called as the *image* of s_0 on C_1. Similarly, we can find the images of s_0 on C_2, C_3 and so on. In the case where s_k is not contained in $P(C_{k+1})$, the image s_{k+1} of s_0 on C_{k+1} is undefined, and the image s_{k+2} of s_0 on C_{k+2} is considered instead. The computation of images of s_0 is terminated when a new image s_{i+1} does not dominate all the cuts before it (i.e., $C_1, \ldots, C_i$). Next, we take s_i as a new starting point and compute the images of s_i on the cuts after it. This procedure is repeatedly performed, until the image on C_m is computed (or considered). In particular, we call the images, which are considered as the starting points, the *critical* images. See Fig. 2 for an example where images s_0 ($= s_6$), s_2, s_4 and s_5 are critical. (Image s_{m+1} is not shown in our figure.)

Let R' denote the route which is a concatenation of the shortest paths between every pair of adjacent critical images s_i and s_j ($0 \leq i < j \leq m + 1$). Clearly, R' is a watchman route, which gives our approximation solution. Let R denote the shortest watchman route through s. An example of routes R' and R is shown in Fig. 2. In the following, we denote by $|xy|$ the length of a line segment $\overline{xy}$, and $|Z|$ the length of a route Z.

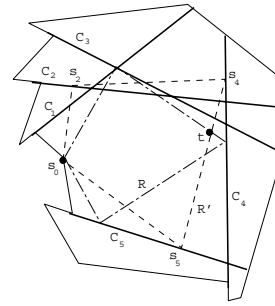


Fig. 2. Critical images and routes R', R.

A watchman route R is said to make a *reflection contact* with cut C if R comes into C at some point and then reflects on C and goes away from that point.

Lemma 2 *If route R reflects on cut C_i and the critical image s_i is defined, then s_i is to the left of the reflection point of R on C_i (as viewed from s_0).*

Proof. Omitted in this extended abstract (see also Fig. 2 for an example). □

Theorem 1 *For any instance of the watchman route problem, $|R'| \leq 2|R|$. Moreover, the watchman route R' can be found in $O(n)$ time, provided that the set of essential cuts is given.*

Proof. Assume first that route R' does not touch the boundary of the given polygon P (except at s). Let s_i and s_j $(1 \leq i < j \leq m)$ denote the first two critical images (excluding s_0). If s_j is contained in $P(C_i)$, it then follows from Lemma 2 that two routes R and R' intersect before s_j. Let t be the intersection point immediately before s_j (along the route R'), and $R_{0,t}$, $R'_{0,t}$ the parts of routes R, R' from s_0 to t, respectively. Let t' denote the point obtained by reflecting t across C_i. Since the chain of segments $\overline{s_0 s_i}$ and $\overline{s_i t'}$ gives the unfolded route $R'_{0,t}$, we have that $|s_0 t'| \leq |R_{0,t}|$. On the other hand, since three points s_0, s_i and t' form an obtuse-angled triangle, it follows from Lemma 1 that $|R'_{0,t}| \leq \sqrt{2}|s_0 t'|$, and thus $|R'_{0,t}| \leq \sqrt{2}|R_{0,t}|$.

Consider the situation where s_j is not contained in $P(C_i)$. Let t be the first intersection of route R' with R, immediately after s_j. (See also Fig. 2 for an example where $s_i = s_2$ and $s_j = s_4$.) Since the next critical image is contained in $P(C_j)$ (otherwise, s_j were not critical), point t is contained in $P(C_j)$, too. Let t' denote the point obtained by reflecting t across C_j. Since s_i, s_j and t' form an obtuse-angled triangle, $|R'_{i,t}| \leq \sqrt{2}|s_i t'|$. Let $R'_{0,t'}$ denote a new route that consists of $\overline{s_0 s_i}$ and $\overline{s_i t'}$. Then we have that $|R'_{0,t'}| \leq \sqrt{2}|s_0 t''|$ and $|s_0 t''| \leq \sqrt{2}|R_{0,t}|$, where t'' is the point obtained by reflecting t' across C_i if t' is contained in $P(C_i)$ (see also Fig. 2), or $t'' = t'$ otherwise. Since $|R'_{0,t'}| = |s_0 s_i| + |s_i t'| \geq |s_0 s_i| + \frac{\sqrt{2}}{2}|R'_{i,t}| > \frac{\sqrt{2}}{2}(|s_0 s_i| + |R'_{i,t})| = \frac{\sqrt{2}}{2}|R'_{0,t}|$, we have that $|R'_{0,t}| < 2|R_{0,t}|$.

The rest parts of R and R' can inductively be considered by taking t as a new starting point. Note that if the rest part of R', say, $R'_{t,m+1}$, contains only one critical image (excluding s_{m+1}), then $|R'_{t,m+1}| \leq \sqrt{2}|R_{t,m+1}|$.

Let us now remove the assumption that R' does not touch the boundary of polygon P. Consider first the case where s_i is the left endpoint of C_i. If the incoming angle of R' with C_i is larger than the outgoing angle (Fig. 3a), the first part of the theorem is still true. Otherwise, it follows from Lemma 2 that s_i should be visited by the route R (Fig. 3b). Then, the first part of the theorem can inductively be proved for two parts of R and R' separated by s_i (i.e., one starts at s_0 and ends at s_i, and the other starts at s_i and ends at s_{m+1}). The other case having to be dealt with is that the shortest path between two critical images, say, s_i and s_{i+1}, wraps around some reflex vertices of P (i.e., the path makes left turns at these vertices when one follows it in the clockwise direction). See Fig. 3c. Let r and r' denote the last and penultimate reflex vertices of P touched by the shortest path between s_i and s_{i+1}. Note that r' is the first point of the shortest path between s_i and s_{i+1}, which is visible to some portion of C_{i+1}. It follows from Lemma 2 that R has to touch r' and/or r. If r is visited by R, the first part of the theorem can inductively be proved for two parts of R' and R, separated by the vertex r. In the situation where only r' is visited by R, the similar treatment can be done for the parts of R and R' from s_0 to r'. Let $R_{r',m+1}$ and $R'_{r',m+1}$ denote the parts of R and R' from r' to s_{m+1}, respectively. Since r is a reflex vertex and s_{i+1} is invisible to r', we extend the segment $\overline{rs_{i+1}}$ until the foot of a perpendicular to the extension and through r' is reached (Fig. 3d). Let $R''_{r',m+1}$ denote the enlarged route. It is easy to see that $|R'_{r',m+1}| < |R''_{r',m+1}| \leq 2|R_{r',m+1}|$. It completes the proof of the first part of the theorem.

Finally, the watchman route R' can be computed in $O(n)$. Due to space limit, we omit the detail. □

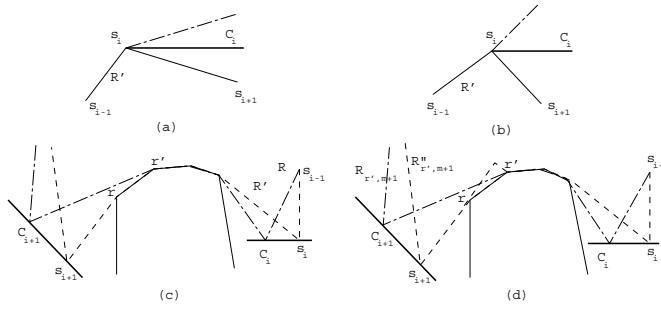


Fig. 3. Illustration for the proof of Theorem 1.

3 Approximating the Shortest Zookeeper's Route

The *zookeeper's* problem is defined as follows: Given a starting point s and a simple polygon (*zoo*) P with a set $\mathcal{P}$ of disjoint convex polygons (*cages*) inside it, each sharing one edge with P, we want to find the shortest route R through s that visits (without entering) at least one point of each cage in $\mathcal{P}$. The best algorithm for computing the shortest zookeeper's route takes $O(n \log^2 n)$ time [1]. Jonsson also gave a 6-approximation solution to the zookeeper's problem [2].

Let $P_1, \ldots, P_m$ denote the cages indexed in a clockwise scan of the boundary of zoo P, starting at s. Also we find a point (the *image*) on the boundary of P_1 that is closest to s_0 in the interior of zoo P (i.e. $P - \mathcal{P}$), a point s_2 on the boundary of P_2 that is closest to s_1, and so on. Putting the shortest paths between s_i and s_{i+1} for all i, $0 \leq i \leq m$, gives a zookeeper's route R'.

Theorem 2 *For any instance of the zookeeper's problem,* $|R'| \leq \sqrt{2}|R|$. *Moreover, the zookeeper's route* R' *can be found in* $O(n)$ *time.*

Proof. Our proof is done by constructing a route R'' such that $|R'| \leq \sqrt{2}|R''|$ and $|R''| \leq |R|$. Let e_i be the edge of cage P_i containing the image s_i. Project all vertices of P_i onto the line containing e_i, and then extend edge e_i in the interior of zoo P rightward (as viewed from s) until the extreme of the projected vertices is reached (or as longer as possible otherwise). See Fig. 4a for an example. Let E denote the set of the line segments, which are obtained by extending all image-defined edges, and let R'' be the shortest route computed by the unfolded method, using the line segments in E as mirrors. (Route R'' may not be a zookeeper's route.) Since R' can also be considered as the route reflecting on all elements of E, it follows the proof of Theorem 1 that $|R'| \leq \sqrt{2}|R''|$.

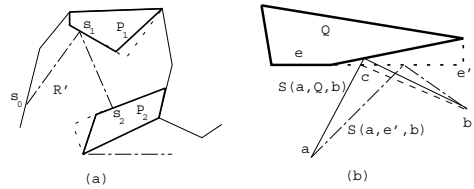


Fig. 4. Illustration for the proof of Theorem 2.

Assume that a and b are two points on the same side of the edge e of a convex polygon Q with $a.x < b.x$ and that the foot of a perpendicular through a is contained in e (Fig. 4b). Let e' denote the line segment obtained by extending edge e rightward until the extreme of the projected vertices of Q is reached, and $S(a, e', b)$ the shortest path visiting a, e' and b in that order. Let $S(a, Q, b)$ denote the shortest path that visits a, some point on the boundary of Q, and b in that order. Let c denote the first intersection of $S(a, Q, b)$ with e'. Clearly, $|S(a, e', b)| \leq |ac| + |cb| \leq |S(a, Q, b)|$ (Fig. 4b). It implies that $|R''| \leq |R|$.

Again, the time taken to compute the route R' is $O(n)$. □

References

1. J.Hershberger and J.Snoeyink, An efficient solution to the zookeeper's problem, *Proc. of the 6th Canadian Conf. on Comput. Geom.*, 104-109, 1994.
2. H.Jonsson, On the zookeeper's problem. In *Proc. 15th Europ. Workshop on Comput. Geom.* (1999) 141-144.
3. X.Tan, T.Hirata and Y.Inagaki, Corrigendum to an incremental algorithm for constructing shortest watchman routes, *Int. J. Comput. Geom. Appl.* **9** (1999) 319-323.

PC-Trees vs. *PQ*-Trees

Wen-Lian Hsu

Institute of Information Science, Academia Sinica, Taipei
hsu@iis.sinica.edu.tw,
http://www.iis.sinica.edu.tw/IASL/hsu/eindex.html

Abstract. A data structure called *PC*-tree is introduced as a generalization of *PQ*-trees. *PC*-trees were originally introduced in a planarity test of Shih and Hsu where they represent partial embeddings of planar graphs. *PQ*-trees were invented by Booth and Lueker to test the consecutive ones property in matrices. The original implementation of the *PQ*-tree algorithms by Booth and Lueker using nine templates in each bottom-up iteration is rather complicated. Also the complexity analysis is rather intricate. We give a very simple *PC*-tree algorithm with the following advantages: (1) it does not use any template; (2) it does all necessary operations at each iteration in one batch and does not involve the cumbersome bottom-up operation. *PC*-trees can be used naturally to test the circular ones property in matrices. And the induced *PQ*-tree algorithm can considerably simplify Booth and Lueker's modification of Lempel, Even and Cederbaum's planarity test.

1. Introduction

A data structure called *PC*-tree is considered here as a generalization of *PQ*-trees. *PC*-trees were originally introduced to represent partial embeddings of planar graphs in Shih and Hsu [7]. *PQ*-trees were used to test the consecutive ones property in matrices [1]. However, the implementation of *PQ*-tree algorithms by Booth and Lueker [1] (hereafter, referred to as the *B&L algorithm*) is rather complicated. Also the complexity analysis is rather intricate. We shall present a very simple *PC*-tree algorithm without using any template. Furthermore, we shall illustrate how to test the circular ones property in matrices using *PC*-trees.

PQ-trees were invented for the more general purpose of representing all permutations of a set U that are consistent with constraints of consecutiveness given by a collection C of subsets of U with the convention that the element of each subset S in C must occur consecutively in the permutation.

The literature on problems related to *PQ*-trees is quite extensive. *PQ*-trees can be applied to test the consecutive ones property of (0,1)-matrices [1,3], to recognize interval graphs [1,2] and to recognize planar graphs efficiently [1,6]. Korte and Möhring [5] considered a modified *PQ*-tree and a simpler incremental update of the tree for the purpose of recognizing interval graphs. Klein and Reif [4] constructed efficient parallel algorithms for manipulating *PQ*-trees. On the other hand, PC-trees were initially used in Shih and Hsu [7] (*S&H*) to represent partial embeddings of planar graphs. Note that this approach is entirely different from Lempel, Even and

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 207–217, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Cederbaum's (*LEC*) approach [6] of using *PQ*-trees to test the consecutive ones property of all nodes adjacent to the incoming node in their vertex addition algorithm. In *S&H*, a P-node is a regular node of the graph, a C-node represents a biconnected component and nodes adjacent to the node in consideration can be scattered anywhere around the *PC*-tree. However, in *LEC*, the leaves of the *PQ*-tree must be those nodes adjacent to the incoming node.

In this paper we shall focus on the application of *PQ*-trees to (0,1)-matrices. A (0,1)-matrix *M* has the *consecutive ones property* (*COP*) for the rows iff its columns can be permuted so that the ones in each row are consecutive. *M* is said to have the *circular ones property* (*CROP*) for the rows if either the ones or the zeros of each row are consecutive. When the zeros of a row are consecutive the ones will wrap around the first and the last column.

A *PQ*-tree is a rooted tree *T* with two types of internal nodes: *P* and *Q*, which will be represented by circles and rectangles, respectively. The leaves of *T* correspond 1-1 with the columns of *M*.

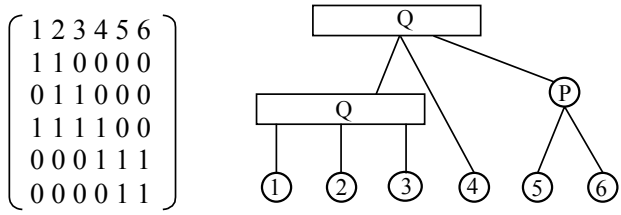


Fig. 1. A *PQ*-tree and the consecutive ones property

We shall follow the notations used in [1]. The frontier of a *PQ*-tree *T*, denoted by *FRONTIER*(*T*), is the permutation of the columns obtained by ordering the leaves of *T* from left to right. Such a permutation is called a *consistent permutation*. The *B&L* algorithm considers each row as a constraint and adds them one at a time. Each time a new row is added, the algorithm tries to modify the current *PQ*-tree to satisfy the consecutiveness constraint of columns of the newly added row. To guarantee unique *PQ*-tree representations we need to restrict ourselves to *proper PQ*-trees defined in [1]: every *P*-node has at least two children; every *Q*-node has at least three children. Two *PQ*-trees are *equivalent* iff one can be transformed into the other by applying zero or more equivalent transformations. The equivalence of two trees is written $T \sim T'$. There are two types of equivalent transformations:

1. Arbitrarily permute the children of a *P*-node,
2. Reverse the order of the children of a *Q*-node.

Denote the set of all consistent permutations (or frontiers) of a *PQ*-tree *T* by $CONSISTENT(T) = \{FRONTIER(T') \mid T' \sim T\}$. Given a *PQ*-tree *T* and a row *u* of *M*, define a new tree called the *u-reduction* of *T* to be one whose consistent permutations are exactly the original permutations in which the leaves in *u* occur consecutively. This new tree is denoted by *REDUCE*(*T*,*u*). Booth and Lueker gave a procedure *REDUCE* to obtain the *u*-reduction. The procedure applies a sequence of *templates* to the nodes of *T*. Each template has a *pattern* and a *replacement*. If a node matches the template's pattern, the pattern is replaced within the tree by the

template's replacement. This is a bottom-up strategy that examines the tree node-by-node obeying the child-before-parent discipline.

A node is said to be *full* if all of its descendants are in u; It is *empty* if none of its descendants are in u; If some but not all of the descendants are in u, it is said to be *partial*. Nodes are said to be *pertinent* if they are either full or partial. The pertinent subtree of T with respect to u, denoted by $PERTINENT(T, u)$, is the subtree of minimum height whose frontier contains all columns in u. The root of the pertinent subtree is denoted by $ROOT(T, u)$, which is usually not the root of the entire tree. There are nine templates used in [1].

In order to achieve optimal efficiency, the algorithm takes several precautions in scanning the pertinent subtree. For example, the maintenance of parent pointers may cause a problem. It is possible that almost all nodes in the tree may receive a new parent even though the row u has only two elements. Therefore, parent pointers are only kept for children of P-nodes and for endmost children of Q-nodes. Interior children of a Q-node do not keep parent pointers. Rather, they borrow them from their endmost siblings. These measures unavoidably complicate the implementation of the B&L algorithm.

A *PC*-tree is a rooted tree T with two types of internal nodes: P and C, which will be represented by circles and double circles, respectively. The leaves of T correspond 1-1 with the columns of M. Figure 2 gives an example of a matrix satisfying the *CROP* (but not the *COP*) and its corresponding *PC*-tree. In obtaining consistent permutations, the children of a P-node can be permuted arbitrarily whereas the children of a C-node observe a circular list, whose order can only be changed from clockwise to counter-clockwise. The $FRONTIER(T)$ of a *PC*-tree consists of those *consistent circular permutations*.

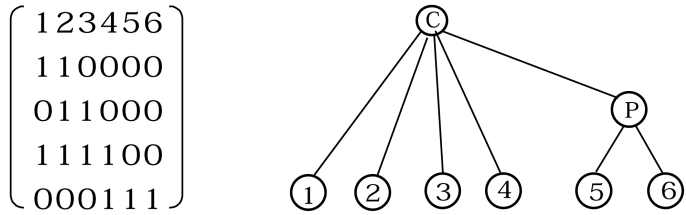


Fig. 2. A *PC*-tree and the circular ones property

For each *PQ*-tree, we can obtain its corresponding *PC*-tree by replacing the Q-nodes with C-nodes. Note that flipping a Q-node in a *PQ*-tree is the same as changing the circular children list of its corresponding C-node from clockwise to counter-clockwise order. There are a few differences between *PC*-trees and *PQ*-trees: (1) When a *PQ*-tree is transformed into a *PC*-tree, and its root is a Q-node, then its two endmost children must be preserved in order to maintain the *COP* in any consistent circular permutation of the transformed *PC*-tree. Aside from this restriction, all other *PC*-tree operations will yield equivalent *PQ*-tree operations. (2) Since it is the circular order of the leaves that needs to be preserved (for example, in Figure 2, we can rotate the tree so that *FRONTIER* becomes 2,3,4,5,6,1), one might as well fix the first column to be 1 in a *PC*-tree in considering any consistent circular permutation. (3)

The root of a *PC*-tree in general is not essential except to maintain the child-parent relationships.

With the additional freedom of rotation, it is natural for *PC*-trees to test the *CROP*. Since our purpose is to illustrate how to replace the complicated bottom-up template matching strategy in *B&L* algorithm, we shall concentrate on testing the *COP* using *PC*-trees.

Our improvement over the B&L algorithm is based on a simple observation that there exist at most two special partial nodes and the unique path connecting them gives a streamline view of the update operation. In particular, no template is necessary in our *PC*-tree algorithm and the node-by-node examination is replaced by *one* swift batch operation.

2. A Forbidden Structure of Consistent Permutations

A key observation that simplifies our *PC*-tree algorithm is based on the following forbidden structure of consistent permutations.

Theorem 1. The following structure $\{S_1, S_2, S_3\}$ is forbidden in any consistent permutation of the columns of M that preserves the *COP*.
Let $S_1 = \{a_1,b_1\}$, $S_2 = \{a_2,b_2\}$ and $S_3 = \{a_3,b_3\}$ be three subsets of columns such that

(1) Columns a_1, a_2 and a_3 are distinct from each other.

(2) None of the a_i's is the same as any of the bi's (but b_1, b_2 and b_3 are not necessarily distinct).

(3) No two columns in any S_i are separated by any column in the other two subsets. Furthermore, b_1, b_2 and b_3 are not separated by any of the a_i's.

Proof. If $b_1 = b_2$, then from (3), we must have the arrangement $a_1 \ldots b_1 (= b_2) \ldots a_2 \ldots$ But then, it would be impossible to place b_3 anywhere in the arrangement unless $b_1 = b_2 = b_3$. In the latter case it would be impossible to place a_3 anywhere in the arrangement.

Hence, assume $b_1 \neq b_2$. From (3), we must have the following arrangement $a_1 \ldots b_1 \ldots b_2 \ldots a_2 \ldots$ (or the reverse) in any consistent permutation of the columns. Since b_1, b_2 and b_3 must not be separated by the a_i's, we shall have $a_1 \ldots b_1 \ldots b_3 \ldots b_2 \ldots a_2 \ldots$ Note that in the above arrangement, b_3 is allowed to be the same as either b_1 or b_2. But then, it would be impossible to place a_3 anywhere in the arrangement.

3. Terminal Nodes and Paths

Define a partial node to be a *terminal node* if none of its children is partial. In other words, each child of a terminal node is either empty or full. Terminal nodes play a major role in simplifying our tree-update operation.

Theorem 2. If the given matrix M satisfies the *COP*, then there can be at most two terminal nodes in T at every iteration.

Proof. Suppose, on the contrary, there are three terminal nodes t_1, t_2, and t_3. Choose an empty child a_i and a full child b_i from each of the t_i, $i = 1, 2, 3$. Then, $S_1 = \{a_1,b_1\}$, $S_2 = \{a_2,b_2\}$ and $S_3 = \{a_3,b_3\}$ constitute a forbidden structure as described in Theorem 1.

If there are two terminal nodes t_1 and t_2, define the unique path connecting t_1 and t_2 to be the *terminal path*. In case there is only one terminal node t, the terminal path is defined to be the unique path from t to $ROOT(T,u)$. In the latter case, t is called a *special terminal node* if it is a *C*-node and its two endmost children are empty. That is, its full children are consecutively arranged in the middle.

Now, any *C*-node in the interior of the terminal path has its children (not on the path) divided into two sides. Since one can flip the children ordering along the terminal path to obtain an equivalent *PC*-tree, we shall show that there is a unique way to flip the children of these *C*-nodes "correctly". Denote $ROOT(T,u)$ by m.

Theorem 3. If M satisfies the *COP* and there are two terminal nodes t_1 and t_2, then
(1) Every *C*-node in the interior of the terminal path satisfies that its full children together with its two neighbors on the path are consecutive in its circular list. Therefore, they can be flipped correctly.
(2) Node m must be on the terminal path.
(3) Let w be a child of a node (other than t) on the unique path from m to the root of T such that w itself is not on this path. Then w must be empty.
Proof. Consider the following cases:

(1) If a *C*-node w does not have any full child, we are done. Hence, assume w has a full child d. Let the two neighbors of w in the terminal path be s_1 and s_2. Suppose, in traversing the circular list of w from s_1 to s_2 through the full child d_1, we encounter an empty child d_2, say in the order s_1, d_1, d_2, s_2. Let d_1', d_2' be two leaves of d_1, d_2, respectively. Choose an empty child a_i and a full child bi from each of the t_i, $i = 1, 2$.

Then $S_2 = \{a_1,b_1\}$, $S_2 = \{a_2,b_2\}$ and $S_3 = \{d_1',d_2'\}$ constitute a forbidden structure as described in Theorem 1.

(2) This is obvious.

(3) Suppose w has a leaf b in u. Since we have $\ldots a_1 \ldots b_1 \ldots b_2 \ldots a_2 \ldots$(or its reverse) in any consistent permutation of the columns, it would be impossible to place b anywhere in the arrangement.

Theorem 4. Consider the case that M satisfies the *COP* and there is only one special terminal node t. Let w be a child of a node (other than m) on the path from m to the root of T such that w itself is not on the path. Then w must be empty.
Proof. Suppose w has a leaf b in u. Let a_1, a_2 be any leaf of the two endmost children of t, respectively and b', a leaf in a full child of t. Then we shall have $a_1 \ldots b' \ldots a_2$ (or its reverse) in any consistent permutation of the columns. But then, it would be impossible to place b anywhere in the arrangement.

By considering the terminal nodes we have a global view of the distribution of all full nodes by Theorems 3 and 4. Figure 3 illustrates an example with two terminal

nodes. Note that, in this example, we have flipped all full children of nodes on the terminal path down.

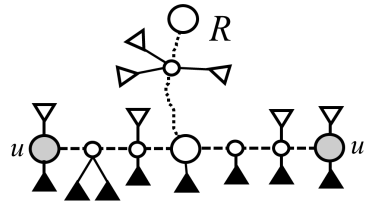


Fig. 3. The unique separating path between two terminal nodes *u* and *u'*

4. Constructing the New *PC*-Tree

Rather than using the node-by-node tree modification in the *B&L* algorithm, we shall update the current *PC*-tree to the new one in a batch fashion.

When a new row *u* is added, perform the following labeling operation:

1. label all leaf columns in *u full*.
2. the first time a node becomes partial or full, report this to its parent.
3. the first time a node *x* gets a partial or full child label *x partial*
4. the first time all children of a node *x* become full label *x full*

Our batch tree-update operation consists of the following steps:

1. delete all edges on the terminal path
2. duplicate the nodes on the terminal path (this is called the *splitting operation*)
3. create a new *C*-node *w* and connect *w* to all duplicated nodes according to their relative positions on the terminal path as follows (this is called the *modifying operation*): connect *w* directly to all *P*-nodes; connect *w* to all full children of *C*-nodes according to their original ordering; contract all degree two nodes

The following figures illustrate the splitting and the modifying operation on the graph of figure 3. Note that the root in a *PC*-tree does not play a major role in these operations.

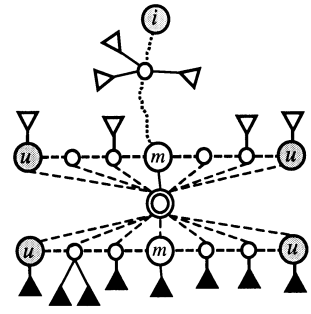


Fig. 4. The splitting operation

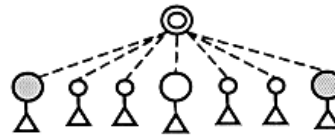


Fig. 5. Connecting to the new C-node

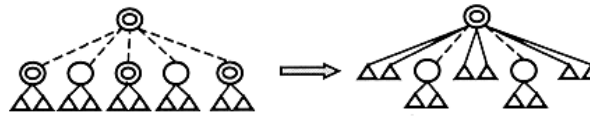
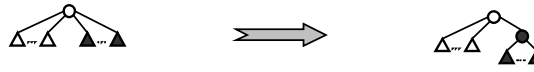


Fig. 6. The modifying operation

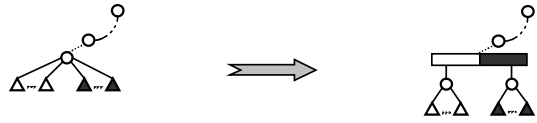
Theorem 5. The corresponding *PQ*-tree of the newly constructed *PC*-tree is the same as the one produced by the *B&L* algorithm.

Proof. As noted before, the *B&L* algorithm modifies the tree in a node-by-node bottom-up fashion based on 9 templates. Whenever a pattern is matched, it is substituted by the replacement.

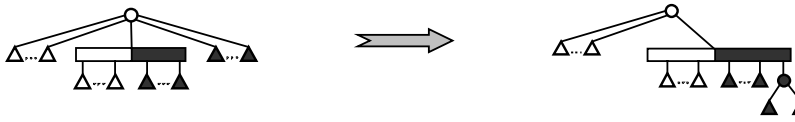
We shall prove the theorem by induction on the depth of the pertinent subtree. For each template operation of the *PQ*-tree, we shall illustrate the corresponding splitting operation of the *PC*-tree and show the equivalence of the two operations in terms of the resulting *PQ*-tree. We shall skip the easy cases of *P*0 and *P*1, *Q*0 and *Q*1.



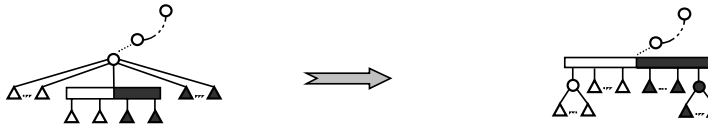
Template *P*2 for *ROOT* (*T,S*) when it is a *P*-node



Template *P*3 for a singly partial *P*-node which is not *ROOT* (*T,S*)



Template *P*4 for *ROOT*(*T,S*) when it is a *P*-node with one partial child (*T,S*)



Template *P*5 for a singly partial *P*-node, other than *ROOT*(*T,S*), with one partial child

Fig. 7. The template operations of Booth and Lurker's algorithm

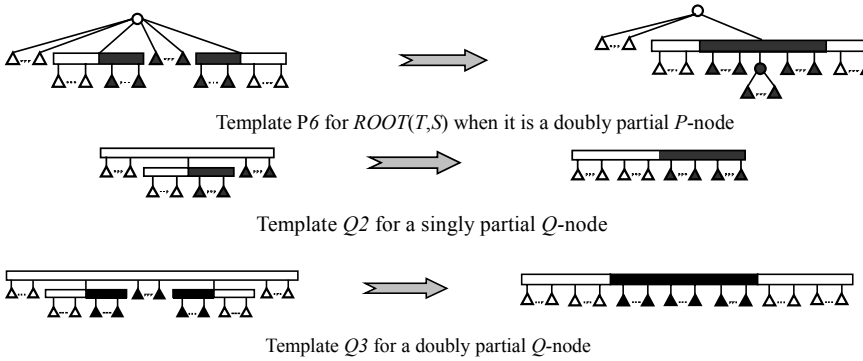


Fig. 7. (Continued)

First, consider templates at the root of *T*. In Figure 8 we consider template P2. The top part of Figure shows the *PQ*-tree replacement. The bottom part gives the corresponding *PC*-tree splitting and modifying operation. The equivalence is shown by the two rightmost diagrams, in which a *PQ*-tree (on top) and its corresponding *PC*-tree (in the bottom) are obtained through *B&L* template matching and the *PC*-tree operation, respectively.

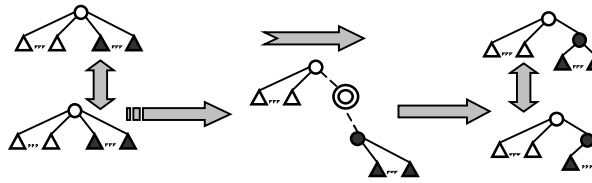


Fig. 8. Template *P*2 for *ROOT* (*T,S*) when it is a *P*-node when it is a *P*-node

The same goes for the templates P4, P6, Q2 and Q3 as shown in Figures 9,10,11,12.

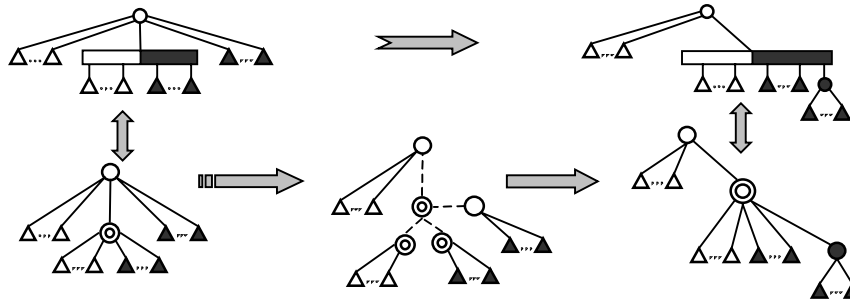


Fig. 9. Template *P*4 for *ROOT*(*T,S*) when it is a *P*-node with one partial child (*T,S*)

Now, assume the theorem is true for *PQ*-trees whose pertinent subtree is of depth k and consider a tree whose pertinent subtree is of depth k+1. In the *PQ*-tree operation, a template will be applied to a non-root terminal node x of the current *PQ*-tree T_1, to obtain a new *PQ*-tree T_2. To simplify the argument we assume the pertinent subtree of T_2 has depth k. Since, by induction, further *PQ*-tree operations on T_2 can be done equivalently through the corresponding *PC*-tree operations, we only have to show that the template matching operation on node x results in the same subtree as the one obtained by operation on the corresponding *PC*-tree of T_2. These are illustrated in Figures 13 and 14.

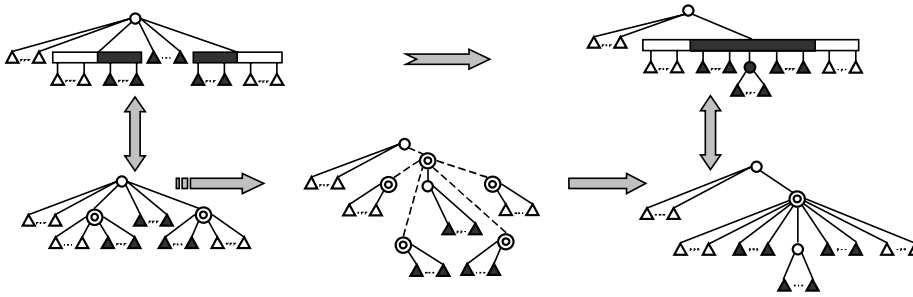


Fig. 10. Template P*6* for *ROOT*(*T*,*S*) when it is a doubly partial *P*-node

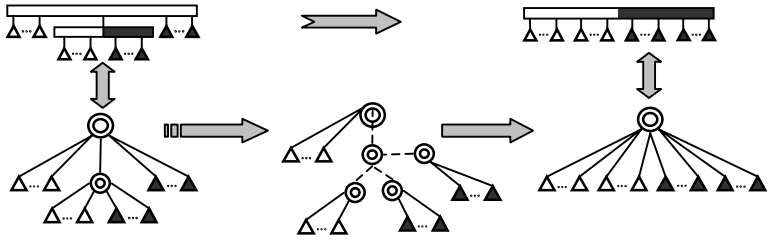


Fig. 11. Template *Q2* for a singly partial *Q*-node

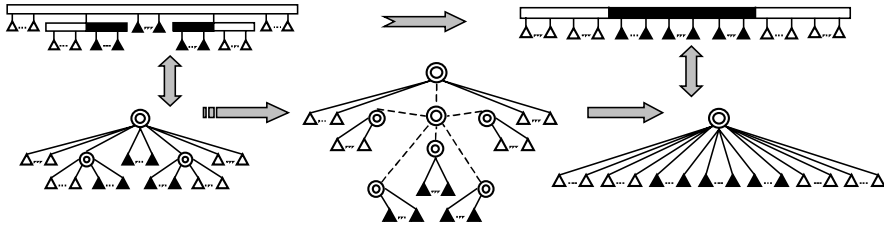


Fig. 12. Template *Q3* for a double partial *Q*-node

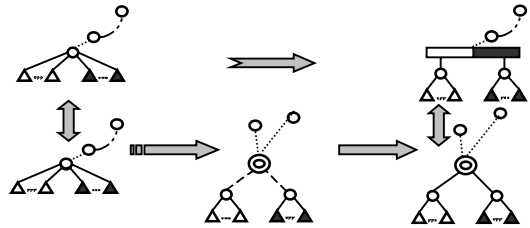


Fig. 13. Template *P*3 for a singly partial *P*-node which is not *ROOT* (*T*,*S*)

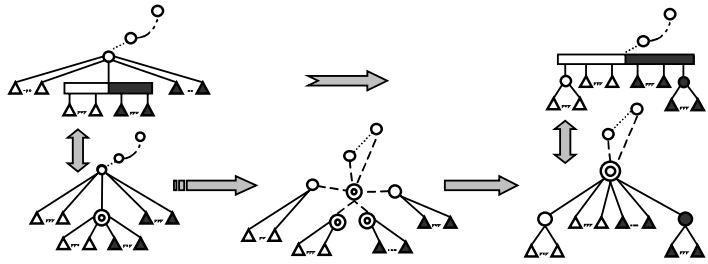


Fig. 14. Template *P*5 for a singly partial *P*-node, other than *ROOT*(*T*,*S*), with one partial child

End of Proof of Theorem 5

5. The Complexity Analysis

In the *B&L* algorithm, the bottom-up propagation potentially could examine the same node many times. Hence, special care must be taken to ensure efficiency. In our splitting and modifying operation of *PC*-trees the update is done in one batch.

We shall apply the same pointer strategy as in the *B&L* algorithm with regard to *P*-nodes and *C*-nodes. Our major saving is in skipping the pattern-matching phase at each node. The remaining replacement phase will be the same as in the B&L algorithm except that we use *C*-nodes instead of *Q*-nodes.

The detailed operations needed for *PC*-trees are all included in those of the *B&L* algorithm. Hence, the running time is linear in the size of the input.

6. Conclusion

We believe that a more efficient implementation of our *PC*-tree algorithm deviating completely from the pointer operation of B&L algorithm is possible. However, the purpose of this paper is to demonstrate that, conceptually, there is a simpler way to view the *PQ*-trees and their updates. Implementation issues of *PC*-trees will be left for those who have the experience of implementing the *B&L* algorithm.

7. Acknowledgement

This research was supported in part by the National Science Council under Grant NSC 89-2213-E-001.

References

1. K.S. Booth and G.S. Lueker, Testing of the Consecutive Ones Property, Interval graphs, and Graph Planarity Using PQ-Tree Algorithms, J. **Comptr. Syst. Sci**. 13, 3 (1976), 335-379.
2. D.R. Fulkerson and O.A. Gross, Incidence Matrices and Interval Graphs, **Pacific Journal of Math**., (1965), 15:835-855.
3. M. C. Golumbic, **Algorithmic Graph Theory and Perfect Graphs,** Academic Press, New York, 1980.
4. P.N. Klein and J.H. Reif, An efficient parallel algorithm for planarity, **J. of Computer and System Science** 37**,** (1988), 190-246.
5. N. Korte and R. H. Möhring, An Incremental Linear-Time Algorithm for Recognizing Interval Graphs, **SIAM J. Comput**. 18, 1989, 68-81.
6. A. Lempel, S. Even and I. Cederbaum, An Algorithm for Planarity Testing of Graphs, **Theory of Graphs**, ed., P. Rosenstiehl, Gordon and Breach, New York, (1967), 215-232.
7. W.K. Shih and W.L. Hsu, Note A new planarity test, **Theoretical Computer Science** 223, (1999), 179-191.

Stacks versus Deques*

Holger Petersen

University of Stuttgart
Institute of Computer Science
Breitwiesenstr. 20–22
D-70565 Stuttgart
petersen@informatik.uni-stuttgart.de

Abstract. We investigate the relative efficiency of a finite number of stacks in comparison to several variants of deques. In the nondeterministic setting, two stacks can simulate a general deque in linear time. This implies a negative answer to the question raised by Brandenburg whether a deque can simulate a finite number of tapes in linear time. We also show that in realtime an output-restricted deque cannot simulate two stacks for deterministic computations. It is known that a general deque can be simulated deterministically by three stacks in linear time. We describe an approach that is simpler to analyze and has a smaller constant factor (with respect to the required stack operations) than a previous solution.

1 Introduction

Functional programming with LISP-like data raises the question whether familiar data structures like queues and deques (double-ended queues) have efficient implementations in the available 'one-ended' lists or stacks. By efficient we mean that the simulator operates in (cumulative) linear time or realtime, i.e., the number of internal operations for each simulated step is bounded by a constant. Notice that the latter is stronger, since it rules out techniques like garbage-collection. On an abstract level, it has been known for a long time that sequential storage with multiple access-pointers—and hence queues and deques—can be simulated by sequential storages with single pointers in linear time [17] and even realtime [5,11]. Since each linear storage can be split into two stacks, this shows the somewhat surprising fact that (up to constant factors) no speed is lost if queues and deques are implemented in functional languages. Concrete implementations of queues have been given by Hood and Melville [8], while Chuang and Goldberg extended this to deques [4]. The known realtime simulations however require a large number of stacks per simulated queue or deque, while there is an obvious way to do a linear time simulation of a single queue by two stacks. It is also known that a deque can be simulated in linear time by three stacks [16]. Questions that arise are: What is the minimum number

* Supported by "Deutsche Akademie der Naturforscher Leopoldina", grant number BMBF-LPD 9901/8-1 of "Bundesministerium für Bildung und Forschung".

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 218–227, 2001.
© Springer-Verlag Berlin Heidelberg 2001

of stacks sufficient for linear or realtime simulation of queue and deque? Does the simulation of a deque become easier if the access to data is restricted?

In the area of formal languages, Ayers [1] investigated automata with a single deque that work in realtime (there called quasi-realtime). In his comments on [1], Brandenburg [3] stated several open problems. He asked whether the hierarchy of language-families accepted by nondeterministic machines working in realtime (or linear time) with a queue, an output-restricted deque, a deque, or a finite number of tapes is proper. These questions have partially been answered. Li e.a. [13] showed that in general even a deterministic machine with a single stack (which obviously can be simulated by an output-restricted deque) cannot be simulated in linear time by a nondeterministic machine with a single queue. An output-restricted deque can be simulated by two stacks in linear time, which by the nondeterministic realtime simulation of many tapes by three stacks [2] and the lower bound of [12, Corollary 3] shows that an output-restricted deque is weaker than a finite number of tapes.

The results presented here concentrate on a small number of stacks and the relation to variants of the deque.

2 Preliminaries

We adopt the terminology from [10]. In our investigations, stacks, queues, and deques are linear lists of atomic items or symbols. Information can be stored and retrieved (along with each deletion) at the ends of the list. In addition, emptiness of the storage can be sensed. More precisely we distinguish between:

Stack: All insertions (push operations) and deletions (pop operations) are made at one end.

Queue: All insertions are made at one end, all deletions at the other end.

Output-restricted deque: Insertions can be made at both ends of the list, while deletions are allowed at one end only.

Input-restricted deque: Deletions can be made at both ends of the list, while insertions are allowed at one end only.

Deque: Insertions and deletions can be made at both ends of the list.

Note that in comparison to [10] we exclude any access to the interior of the lists[1].

The usual framework in which these data storages are compared is by attaching them to a finite control that has a separate read-only one-way input-tape and either accepts or rejects an input, see e.g. [15,13].

3 Nondeterministic Simulation of a Deque by Two Stacks

We first consider nondeterministic machines and show that—up to a linear time overhead—machines with a single deque are equivalent in accepting power to those equipped with two stacks.

[1] What is called stack here is also known as a pushdown-store. The name 'stack' is often used for a storage that admits read access to its interior, which we don't allow.

Theorem 1. *Every $t(n)$ time-bounded nondeterministic deque machine can be simulated by an $O(t(n))$ time-bounded nondeterministic two-stack machine.*

Proof. We will divide the simulation into two stages. The first stage simulates in realtime a deque by a queue and two stacks S_L and S_R with some restrictions on the way the queue can be accessed. In the second stage we show how to simulate a queue and two stacks with two stacks within the time-bound claimed in the theorem.

Without loss of generality we assume that the deque is empty when the computation terminates. We claim that a single deque can be simulated by a queue and two stacks step by step as follows. The simulator starts with empty stacks and an empty queue. It reads input and executes operations in accordance to the finite control of the deque machine. In addition the simulator is in one of two modes, either in left-right or in right-left mode (these names will be explained later). The initial mode can be chosen arbitrarily and during the simulation the simulator may switch from one mode to the other whenever the queue is empty. The possible deque operations are now simulated in the following way:

Insert at the left end: Nondeterministically choose to push the new symbol onto S_L or, provided S_L is empty and the current mode is left-right, insert the new symbol into the queue.

Insert at the right end: Nondeterministically choose either to push the new symbol onto S_R or, provided S_R is empty and the current mode is right-left, insert the new symbol into the queue.

Delete at the left end: If S_L is not empty, pop one symbol. If S_L is empty, remove one symbol from the queue, provided it is not empty and the current mode is right-left. Otherwise abort the simulation.

Delete at the right end: If S_R is not empty, pop one symbol. If S_R is empty, remove one symbol from the queue, provided it is not empty and the current mode is left-right. Otherwise abort the simulation.

Sense emptiness: Check that S_L, S_R, and the queue are empty.

The main invariant of the simulation (which holds after the simulation of one deque access) is, that the current deque contents of the simulated machine are always encoded by the concatenation of the contents of S_L with its top to the left, the queue either with its rear to the left (in left-right mode) or to the right (in right-left mode), and S_R with its top to the right. By checking the simulations of the four operations on the deque and their subcases it can be verified that this invariant is maintained and that the access is always to the symbols at the encoding of the ends of the deque. We thus see that each successful run of the simulator corresponds to an accepting computation of the deque machine.

Conversely, we have to argue that every accepting computation of the deque machine can be simulated successfully. Fix an accepting computation of the deque machine. We can associate with every symbol stored on the deque (which we assume to be distinguishable for our considerations) two bits of information: whether it entered the deque from the left or the right end and whether it will leave the deque at the left or the right end (we assumed that the deque

will eventually be empty). The symbols will be sorted according to each of the components, which leads to the important observation that in each configuration of the deque there is at most one type of symbols with a 'mixed mode' (left-right or right-left). Let us assign to each configuration of the deque the 'mixed mode' of its contents (or an arbitrary mode if there is no symbol with mixed mode). The simulator now follows the information associated with the symbols and always works in the mode prescribed by the mode of the configuration of the deque. It does so by storing the left-left symbols on S_L, the right-right symbols on S_R, and all mixed-mode symbols (left-right or right-left) on the queue. Clearly the mode of the deque can only change when there is no mixed-mode symbol, and then the queue of the simulator is empty allowing it to switch between the modes as well. If the deque contains symbols that should be deleted at the left end, the simulator will either store them on S_L, where they are readily available, or they are mixed-mode symbols, which forces them to be at the front of the queue where they can be accessed by the simulator. An analogous argument shows that deletions at the right end can be carried out faithfully.

By the above discussion each computation of the deque machine can be simulated and each run of the simulator corresponds to a computation of the deque machine.

Now we show how to simulate the two stacks and one queue on two stacks. The simulated stacks will be mapped onto the corresponding stacks of the simulator, with the additional property that there is a bottom marker below each portion encoding a simulated stack. Below the bottom markers two strings are stored, where the shorter string is a suffix of the longer if the simulation is to be completed successfully (if both strings have the same length, they should be identical). If the simulated machine is in left-right mode, then the string below the marker stored on S_L is at least as long as the one on S_R and vice versa for right-left mode. If a symbol is to be entered into the queue, then this symbol is added to the 'longer' string, if a symbol is deleted it is guessed and added to the shorter string. Note that accessing the strings below the marker is always possible since the corresponding stack will be empty at that point of the simulation.

If eventually an accepting state has been reached, the contents of the two stacks are compared symbol by symbol in $O(t(n))$ time and the input is accepted if they are identical.

The simulation strategy described so far admits a simulation of each valid computation. It may however include invalid computations as well for two reasons. First, switching from one mode to the other is allowed only if the queue is empty, and the simulator has to verify this condition. Second, even if switching from one mode to the other occurs only at valid points during the simulation, symbols might be deleted from the simulated queue before they have been inserted. The first problem is overcome by inserting a special zero marker into the guessed strings after the last symbol has been entered resp. deleted just before switching the mode. Consider a situation when the simulator is in left-right mode. Then after the last symbol during this phase has been stored on S_R the

simulator adds the zero marker to the string on S_R and records that the marker has been set. Note that above this zero marker the stack can still be simulated, but further insertions are not allowed until another zero marker is stored on S_L and the mode changes to right-left. The simulator works analogously in right-left mode. Zero markers necessarily occur in pairs, one on each stack of the simulator. Therefore the length of the strings at steps when the mode changes are equal (assuming that the final comparison is successful) and the simulated queue is empty at these points.

For the second problem we associate with each symbol a flag from the set $\{0, 1\}$ and we let the simulator remember the current flag f. With each symbol inserted into the queue, the simulator associates $f + 1 \pmod 2$, with every deleted symbol the flag f. If the current value of the flag has been used for at least one insertion it may nondeterministically be incremented by 1 modulo 2. Assume for a contradiction that the resulting strings could be equal although at least one symbol has been deleted from the queue being simulated before it was inserted. Consider the situation just before this happens for the first time. The lengths of the strings will be identical and the flags of the topmost pair of symbols match, both having the value f (since we assumed that the strings would eventually be equal). The flag has been incremented since the last insertion without having been used for another one. No further modification of the flag is possible before the next symbol is inserted into the simulated queue. Therefore the (erroneously) deleted symbol gets flag f as well. The next symbol to be inserted will have flag $f + 1 \pmod 2$, which will cause a mismatch, contradicting our assumption.

Since each step in the two stages of the simulation can be carried out in constant time, we see that the entire simulation can be completed in $O(t(n))$ steps. □

Together with the obvious simulation of two stacks by a deque (the stacks are stored with bottom markers next to each others at both ends of the deque) we obtain:

Corollary 1. *For nondeterministic machines a single deque and two stacks are equivalent up to a linear time overhead.*

We can now answer the question from [3] whether nondeterministic machines with several tapes working in linear time are stronger than linear-time machines with one deque. Informally we obtain that two tapes (which are equivalent to many tapes [2]) are stronger than a single deque.

Corollary 2. *There is no general linear time simulation of multiple tapes by one deque for nondeterministic machines.*

Proof. Suppose that a linear time simulation of two tapes by a deque exists. For the special case of realtime acceptance by two tapes we obtain an upper bound of $O(n)$ on two stacks. Simulating these two stacks by the method of [12] on a single tape we obtain an $O(n^{1.5}\sqrt{\log n})$ upper bound on the time. But this contradicts the known almost quadratic lower bounds for the nondeterministic simulation of two tapes by one tape, e.g. $\Omega(n^2/\log^{(k)} n)$ for any positive k from [6] ($\log^{(k)}$ denotes the k-fold iteration of log). □

By calculating the simulation overhead we can even derive the lower bound $\Omega(n^{4/3}/\sqrt{\log n})$ for the nondeterministic simulation of two tapes by one deque.

For nondeterministic computations three stacks are as powerful as any finite number of tapes [2]. Simulations by three or more stacks are therefore interesting only in the deterministic setting. In the next section we will consider some simulations of this kind.

4 Deterministic Simulations

For deterministic simulations of a deque by two stacks we have a less complete picture than in the nondeterministic setting. The well-known linear time simulation of a queue by two stacks (see, e.g., [8,13] or [10, Ex. 2.2.1.14]) generalizes to output-restricted deques, while we don't have an analogous result for input-restricted or general deques.

The question whether there is a realtime equivalence between two stacks and an output-restricted deque is answered negatively by the following results.

Let u^R denote the reversal of string u. Define $L \subseteq \{0,1,2,3,\$\}^*$ as

$$L = \{w_1\$w_2\$w_3xy\$z \mid \quad w_1, w_2, w_3 \in \{0,1\}^*, x \in \{2,3\},$$
$$y = w_2^R \text{ if } x = 2, \quad y = w_3^R \text{ if } x = 3, \quad \exists t \in \{0,1\}^* : w_1 = tz^R\}.$$

Lemma 1. *There is a deterministic two-stack machine that accepts L in realtime.*

Proof. Clearly, the machine can decide membership in the regular language $\{0,1\}^*\$\{0,1\}^*\$\{0,1\}^*\{2,3\}\{0,1\}^*\$\{0,1\}^*$ on the fly. We may therefore assume that the input can be decomposed according to the definition of L. The machine accepting L pushes copies of w_1 on *both* of its stacks and marks these segments of its stacks. Then it copies w_2 onto stack 1 and w_3 onto stack 2. When x is encountered it accordingly compares the string stored on stack 1 or stack 2 with the input segment y following x. If y matches, the remaining symbols of the input after the last \$ are compared with the topmost section of the stack which has just been emptied (discarding the marker first). □

Lemma 2. *Deterministic machines with one output-restricted deque cannot accept L in realtime.*

Proof. We assume that L can be accepted by a machine M of the kind described in the lemma (with a deque that allows deletions only at its rear) and derive a contradiction. We simplify the proof by assuming that M executes one step per input symbol, which can be achieved by possibly increasing the number of internal states and of storage symbols.

The proof uses the notion of incompressibility, in particular the non-constructive existence of incompressible binary strings of each length, see [14]. We will also require some strings of symbols to be self delimiting, which means that

their last symbols can be detected even if they are concatenated with other strings. A simple (though not very efficient) way to achieve this is to repeat each bit except for the last one, which is encoded by one of 01 or 10.

Let m be the number of different symbols available to M for storing information on its deque. Fix the constant $c = \max(2, 3\lceil \log_2 m \rceil)$. Choose an incompressible string w of length $(c^2 + 2c + 1)n$ for n sufficiently large to make the argument work. Divide w into three portions $w = w_1 w_2 w_3$ with $|w_1| = (c^2 + c)n$, $|w_2| = n$, and $|w_3| = cn$. We now consider accepting computations of M on input strings with a prefix $w_1\$$. Our first claim is that after having read this prefix the length ℓ of M's deque exceeds $2(c+1)n$. We can recover w_1 from the contents of the deque by feeding the string $\$2\$$ into M (this will encode three empty strings w_2, w_3, and y) and then looking symbol by symbol for the longest trailing string that causes M to accept. Since any string that leads to acceptance is a prefix of w_1^R, this will recover w_1^R. Now suppose $\ell \leq 2(c + 1)n$. The entire string w can be described by:

- A self delimiting description of M of size $O(1)$.
- The state of M after reading $w_1\$$ of size $O(1)$.
- A self delimiting description of the above algorithm for recovering w_1 of size $O(1)$.
- A self delimiting encoding of ℓ in $O(\log n)$ bits.
- A self delimiting encoding of n in $O(\log n)$ bits.
- The contents of M's deque encoded in binary in at most $\ell\lceil \log_2 m \rceil \leq \frac{2}{3}(c^2 + c)n$ bits.
- The strings w_2 and w_3 (no encoding is necessary, since their lengths are known).

For sufficiently large n this will result in a compression of w contradicting its choice.

Notice that after reading $w_1\$$ the deque is now longer than the remaining suffix of any accepted input of the form we consider with an empty trailing portion z. This implies that the first symbol on M's deque at this point will never be removed and nothing added at the head can later influence the computation.

Consider next the state of M's deque when $w_1\$w_2\$w_3$ has been read. We count the number of trailing positions of these deque contents that have been modified while M was reading w_3. Assume first that this number exceeds $n+1$. Intuitively, M has—up to a finite number of bits—hidden (or even destroyed) the encoding of w_2. If the trailing $n + 2$ symbols on the deque are known, then w_2 can be recovered by starting M in its state q_1 after reading $w_1\$w_2\$w_3$ and feeding in strings of the form $2u\$$ with $u \in \{0, 1\}^n$ until the unique string $2w_2^R\$$ leading to acceptance has been found. The trailing segment of length $n + 2$ of the deque can be generated by starting M from the state q_2 it was in before it accessed the first symbol in this segment for the last time and feeding in the portion of w_3 that had not been processed at this point.

We can now describe w by:

- A self delimiting description of M of size $O(1)$.

- The states q_1 and q_2 in $O(1)$ bits.
- A self delimiting description of the above algorithm for recovering w_2 of size $O(1)$.
- A self delimiting encoding of n in $O(\log n)$ bits.
- A self delimiting encoding of the position within w_3 from which M has to be started in order to generate the trailing portion of the deque in $O(\log n)$ bits.
- The strings w_1 and w_3 (again, their length is available).

For n sufficiently large we obtain a compression contradicting the choice of w.

The other case is where a trailing string u of at most $n+1$ symbols of the deque includes all positions modified while reading w_3. It is dealt with in a similar way, now compressing w_3. □

As a consequence of the two preceding lemmas we obtain:

Theorem 2. *Deterministic machines with one output-restricted deque cannot simulate two-stack machines (and hence deque machines) in realtime.*

Two stacks cannot recognize the language $\{x2x' \mid x \in \{0,1\}^* \text{ and } x' \text{ is a prefix of } x\}$ by the (stronger) result of [9] in realtime, while even with a queue this language can be accepted. Therefore two stacks and an output-restricted queue are of incomparable power for deterministic realtime computations.

Next we consider simulations by more than two stacks. Rosenberg [16] describes a linear time simulation of a general deque by three stacks. The idea behind the simulation is again to maintain a left and a right section of the deque on two stacks. Whenever exactly one of these sections is empty and a delete operation on the respective portion is requested, the third stack is used in order to partition the non-empty section into segments of approximately half of its original size. The analysis of this simulation is somewhat difficult, since items may be copied from one section to the other and back an unbounded number of times.

We present here a different linear time simulation of a deque by three stacks, which is easier to analyze and has a smaller linear factor in terms of the number of push and pop operations required. While Rosenberg's solution uses at most $18n$ stack-operations for simulating n deque-operations, our bound will be $9n$. We remark that Rosenberg's simulation is closely related to the construction used by Stoss [17] in the simulation of multi-head tape-units by single-head ones, while ours is inspired by the very first proof of such a simulation given by Hartmanis and Stearns [7].

Theorem 3. *A sequence of n deque operations can be simulated deterministically by three stacks with $9n$ stack-operations.*

Proof. Like in Rosenberg's simulation, we represent a deque as two stacks S_L and S_R that encode the current contents of the deque. In contrast to the simulation outlined above, where (except for intermediate states) the concatenation of the two stacks reflects the deque contents verbatim, in our simulation some information may be duplicated or outdated. Therefore some deletions at the bottom

of the stacks might be necessary in order to obtain the current contents of the simulated deque. The membership of an item in the section possibly outdated or duplicated will be expressed by coloring it red, while all other items will be green. These colors can either be represented by a single bit attached to each item, or by markers on the stacks, since items of the same color form blocks. Initially S_L and S_R are empty, as is the third stack S_T that will hold a counter which has the value 0 at the start of the simulation. Insert operations are simply mapped to push operations of green items onto S_L or S_R. We will now describe a delete operation on the left in detail. The corresponding operation on the right works symmetrically.

If S_L contains an item that is colored green as its topmost item, then it is returned. If S_L has a red item as its topmost and the count maintained on the third stack is not 0, then the red item is returned and the counter is decremented. Otherwise a clean-up phase is initiated. All remaining items on S_L are deleted. The green items on top of S_R are popped one after the other and pushed onto S_L as well as onto S_T, changing their color to red. The remaining red items on S_R are discarded. The items now on S_T are moved back onto S_R and their number is counted on S_L above its red items. Finally the count is transferred to S_T. Then the simulation is resumed and the deque operation is carried out, if possible.

In order to see that the simulation is correct, note that the current contents of the deque being simulated is encoded by the green items on S_L (top to bottom), a top segment of c red items on S_L (where c is the value currently stored on S_T), or equivalently the mirror image of a top segment of red items on S_R, and finally the green items on S_R from bottom to top. This invariant is clearly maintained by push operations and pop operations of green items. A pop operation of a red item reduces the count and thus adapts the length of the red segments considered on both stacks. The clean-up moves a sequence of green items into fully overlapping red segments.

Each item enters in green color, causing one push (1 operation). If it leaves the deque while still having green color, the total cost is 2 operations. Otherwise it changes its color to red. While doing so, it is moved from S_L or S_R to the other stacks (3 operations), one of the copies is moved back from S_T and counted (3 operations), and the count is transferred to S_T (2 operations). If it is never removed, then both copies stay in the stacks and the total cost is 9 operations. Otherwise the item is removed and possibly also its other copy (2 operations). In addition the count is updated (1 operation). In the latter case the total is at most 12 push or pop operations for simulating two operations, which yields 6 operations per simulated deque operation. □

In the simulation given in the previous proof red items can be left on the stacks in the clean-up phase (which then doesn't really clean-up), saving n operations at the cost of wasting stack space. Also, the counter values could be reduced by a linear factor, which would save $(3 - \epsilon)n$ operations for any $\epsilon > 0$, an improvement which can be applied to Rosenberg's simulation as well.

Acknowledgments

Thanks are due to Amir M. Ben-Amram for several helpful remarks. I am very grateful for support by the Academic College of Tel-Aviv-Yaffo, where the research reported here was done.

References

1. Kathleen Ayers. Deque automata and a subfamily of context-sensitive languages which contains all semilinear bounded languages. *Theoretical Computer Science*, 40:163–174, 1985.
2. Ronald V. Book and Sheila A. Greibach. Quasi-realtime languages. *Mathematical Systems Theory*, 4:97–111, 1970.
3. Franz J. Brandenburg. A note on: 'Deque automata and a subfamily of context-sensitive languages which contains all semilinear bounded languages' (by K. Ayers). *Theoretical Computer Science*, 52:341–342, 1987.
4. Tyng-Ruey Chuang and Benjamin Goldberg. Real-time deques, multihead Turing machines, and purely functional programming. In *Conference on Functional Programming Languages and Computer Architecture*, pages 289–298, 1993.
5. Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Real-time simulation of multihead tape units. *Journal of the Association for Computing Machinery*, 19:590–607, 1972.
6. Zvi Galil, Ravi Kannan, and Endre Szemeredi. On nontrivial separators for k-page graphs and simulations by nondeterministic one-tape Turing machines. In *Proceedings of the 18th ACM Symposium on Theory of Computing (STOC), Berkeley, California*, pages 39–49, 1986.
7. Juris Hartmanis and Richard E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.
8. Robert Hood and Robert Melville. Real time queue operations in pure Lisp. *Information Processing Letters*, 13:50–54, 1981.
9. Tao Jiang, Joel I. Seiferas, and Paul M. B. Vitányi. Two heads are better than two tapes. *Journal of the Association for Computing Machinery*, 44:237–256, 1997.
10. Donald E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Reading Mass., 3rd edition, 1997.
11. Benton L. Leong and Joel I. Seiferas. New real-time simulations of multihead tape units. *Journal of the Association for Computing Machinery*, 28:166–180, 1981.
12. Ming Li. Simulating two pushdown stores by one tape in $O(n^{1.5}\sqrt{\log n})$ time. *Journal of Computer and System Sciences*, 37:101–116, 1988.
13. Ming Li, Luc Longpré, and Paul Vitányi. The power of the queue. *SIAM Journal on Computing*, 21:697–712, 1992.
14. Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer, Berlin-Heidelberg-New York, 1993.
15. Ming Li and Paul M. B. Vitányi. Tape versus queue and stacks: The lower bounds. *Information and Computation*, 78:56–85, 1988.
16. Burton Rosenberg. Simulating a stack by queues. In *Proceedings of the XIX Latinamerican Conference on Computer Science*, volume 1, pages 3–13, 1993.
17. Hanns-Jörg Stoß. k-Band Simulation von k-Kopf Turingmaschinen. (k-tape simulation of k-head Turing machines). *Computing*, 6:309–317, 1970. In German.

Optimizing a Computational Method for Length Lower Bounds for Reflecting Sequences

H.K. Dai

Computer Science Department, Oklahoma State University,
Stillwater, Oklahoma 74078, U. S. A.
dai@cs.okstate.edu

Abstract. We refine and optimize a computationally intensive enumeration method, based on the traversal of a quadtree, for finding lower bounds on the lengths of reflecting sequences for labeled chains. The improvement results from the introduction of a redundancy relation defined on vertex-pairs of the underlying quadtree, which enables the pruning of redundant branches near the root of the quadtree, as well as locally at deeper depths. The test run of the implementation showed a length lower bound of $19t-214$ for t-reflecting sequences for labeled 7-chains with significant speedup, which yields the current length lower bound $\Omega(n^{1.51})$ for universal traversal sequences for 2-regular graphs of n vertices, and $\Omega(d^{2-1.51}n^{2.51})$ for universal traversal sequences for d-regular graphs of n vertices, where $3 \leq d \leq \frac{n}{17} + 1$.

1 Preliminaries

Reflecting sequences are variants of universal traversal sequences, and were introduced by Tompa [9] in proving lower bounds on the lengths of universal traversal sequences. The study of universal traversal sequences is motivated by the complexity of graph traversal. Good bounds on the lengths of universal traversal sequences translate into good bounds on the time complexity of certain undirected graph traversal algorithms running in very limited space.

Aleliunas et al. [1] proved the existence of polynomial-length universal traversal sequences, yielding polynomial-time and logarithmic-space (non-uniform) deterministic algorithms for undirected graph traversal. Beame et al. [2] used variants of the jumping automaton for graphs of Cook and Rackoff [6] to study time-space tradeoffs for undirected graph traversal. Good lower bounds on the length of universal traversal sequences provide a prerequisite to proving time-space tradeoffs for traversing undirected graphs for these computational models.

1.1 Universal Traversal Sequences for Undirected Graphs

For a graph G, denote by $V(G)$ the vertex set of G, and by $E(G)$ the edge set of G. For positive integers d and n such that $d < n$, let $\mathcal{G}(d, n)$ be the set of all connected d-regular graphs of order n with an edge-labeling. The edge-labeling on $G \in \mathcal{G}(d, n)$ is defined as follows: For every edge $\{u, v\} \in E(G)$, there are two

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 228–236, 2001.
© Springer-Verlag Berlin Heidelberg 2001

labels $l_{u,v}, l_{v,u} \in \{0, 1, \ldots, d-1\}$ with the property that for every $u \in V(G)$, $\{l_{u,v} \mid \{u, v\} \in E(G)\} = \{0, 1, \ldots, d-1\}$. Note that $\mathcal{G}(d, n)$ is not empty if and only if dn is even. A sequence $U \in \{0, 1, \ldots, d-1\}^*$ can be considered as a sequence of edge-traversal commands and induces a walk starting at any vertex in every labeled graph in $\mathcal{G}(d, n)$. We say that such a sequence U traverses $G \in \mathcal{G}(d, n)$ starting at $v_0 \in V(G)$ if, by starting at v_0 and following the sequence of edge labels in U, all the vertices in G are eventually visited. The sequence U is called a *universal traversal sequence* (UTS) for $\mathcal{G}(d, n)$ if U traverses every $G \in \mathcal{G}(d, n)$ starting at every vertex in G.

Let $U(d, n)$ denote the length of a shortest UTS for non-empty $\mathcal{G}(d, n)$, and define $U(d, n) = U(d, n+1)$ in case $\mathcal{G}(d, n)$ is empty. The lower and upper bounds on $U(d, n)$ for various ranges of d were studied in [1], [3], [4], [7], [9], [5], and [8]. Currently, the best lower bounds on $U(d, n)$ are in [8],

$$U(d,n) = \begin{cases} \Omega(n^{\log_7 19}) & \text{if } d = 2\ , \\ \Omega(d^{2-\log_7 19} n^{1+\log_7 19}) & \text{if } 3 \le d \le \frac{n}{17} + 1\ . \end{cases}$$

which were obtained using a computationally intensive approach.

1.2 Reflecting Sequences for Labeled Chains

For positive integer n, a *labeled chain* of length n is a graph G with vertex set $V(G) = \{0, 1, \ldots, n\}$ and edge set $E(G) = \{\{i, i+1\} \mid 0 \le i < n\}$ with an edge-labeling defined as follows: Every edge $\{i, i+1\} \in E(G)$ is associated with two labels $l_{i,i+1}$ and $l_{i+1,i}$, each a non-empty subset of $\{0, 1\}$, such that

1. $l_{0,1} = l_{n,n-1} = \{0, 1\}$, and
2. $l_{i,i-1}$ and $l_{i,i+1}$ form a partition of $\{0, 1\}$ for all $0 < i < n$.

Denote by $\mathcal{L}(n)$ the set of all labeled chains of length n. A labeled chain $G \in \mathcal{L}(n)$ can be identified with the sequence $\alpha = \alpha_1 \alpha_2 \cdots \alpha_{n-1} \in \{0, 1\}^{n-1}$, where $l_{i,i+1} = \{\alpha_i\}$ for $0 < i < n$.

Given a labeled chain $G \in \mathcal{L}(n)$, every sequence $U = U_1 U_2 \cdots U_k \in \{0, 1\}^k$ where $k \ge 0$, when considered as an edge-traversal sequence starting at vertex 0 in G, determines a unique sequence $(0 = v_0, v_1, v_2, \ldots, v_k) \in \{0, 1, \ldots, n\}^{k+1}$ such that $U_i \in l_{v_{i-1}, v_i}$ for all $1 \le i \le k$. For positive integer t, a sequence $U \in \{0, 1\}^*$ is said to *reflect t times* in $G \in \mathcal{L}(n)$ if the walk in G induced by U alternately visits the end-vertices n and 0 at least t times; that is, there exist $0 < i_1 < i_2 < \cdots < i_t \le |U|$ such that $v_{i_{2j-1}} = n$ for all $1 \le j \le \lceil \frac{t}{2} \rceil$ and $v_{i_{2j}} = 0$ for all $1 \le j \le \lfloor \frac{t}{2} \rfloor$. The sequence U is called a *t-reflecting sequence* for $\mathcal{L}(n)$ if U reflects t times in every $G \in \mathcal{L}(n)$. Denote by $R(t, n)$ the length of a shortest t-reflecting sequence for $\mathcal{L}(n)$.

The following theorem reduces the length lower bounds for universal traversal sequences to those for t-reflecting sequences.

Theorem 1. *[8] Suppose that there exist positive integer $c \ge 2$ and positive reals $r \ge 2$ and k such that, for every positive integer t, $R(t, c) \ge rt - k$. Then*

$$U(d,n) = \begin{cases} \Omega(n^{\log_c r}) & \text{if } d = 2\ , \\ \Omega(d^{2-\log_c r} n^{1+\log_c r}) & \text{if } 3 \le d \le \frac{n}{17} + 1\ . \end{cases}$$

In the light of Theorem 1, we can obtain length lower bounds for universal traversal sequences by seeking constants c, r, and k such that every t-reflecting sequence for $\mathcal{L}(c)$ has length at least $rt-k$, in which a large $\log_c r$ translates into a good length lower bound. The lemma below shows that it suffices to consider only the case when t is odd.

Lemma 1. *[8] For all positive integers t and c and positive reals r and k, if $R(2t-1,c) \geq 2rt-k$ for every positive integer t, then $R(t',c) \geq rt'-k$ for every positive integer t'.*

2 The Quadtree Enumeration Method

To show that $R(2t-1,c) \geq 2rt-k$ in Lemma 1, we assign a minimum of $2^{c-1}t$ "marks" to various positions in a hypothetical $(2t-1)$-reflecting sequence S (t marks for each labeled chain of length c). We argue that the "marks-to-bits" density for all but a short suffix of S must be at most $\frac{2^{c-1}}{2r}$, implying a minimum length of $2^{c-1}t/(2^{c-1}/2r) = 2rt$ for all but a short suffix of S having a bounded number of marks. We then determine k, depending on the maximum number of marks in the suffix, which gives that $R(2t-1,c) \geq 2rt-k$.

Following the framework developed and analyzed in [8], it suffices to construct a tree T that can be used to divide every marked S into even-length segments, such that all but possibly the last segment has a marks-to-pairs density at most $\frac{2^{c-1}}{r}$. Specifically, T is a quadtree whose vertices represent potential even bit-positions in a hypothetical $(2t-1)$-reflecting sequence for $\mathcal{L}(c)$, and whose branches from a position are labeled by the four possible pair-continuations at that position, that is, 00, 01, 10 and 11.

We summarize the Quadtree Enumeration method [8] in the following algorithm.

algorithm Build_Quadtree (c, ρ)
1. Initialize S to the pair 00;
2. (* Only examine 00-branch by symmetry. *)
   While $S \neq 01$ do
   2.1. Try to find a prefix (pair-sequence) of S whose marks-to-pairs density is at most ρ;
   2.2. If no such prefix exists, then
      2.2.1. Extend the quadtree by appending the pair 00 to S;
   else
      2.2.2. (* The sequence S represents a leaf. *)
         Replace S with the pair-sequence no longer than S and following S in the lexicographic ordering of pair-sequences;

end Build_Quadtree

To determine in step 2.1 an upper bound on the marks-to-pairs density of a sequence S, we assume the maximum number of marks possible for each chain in $\mathcal{L}(c)$, which can be obtained by assuming the traversal of the chain starting at vertex 1 with the traversal sequence S. Hence, for each $C_i \in \mathcal{L}(c)$, where $i \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$, we traverse C_i starting at vertex 1 with S, and place two types of marks M_i on the pairs of S as follows.

1. *Closed marks* of M_i on S that correspond to the last exit from vertex 1 during every traversal from vertex 0 to vertex c with S, and

2. An *open mark* of M_i on S that corresponds to the last exit from vertex 1 during a traversal from vertex 0 without visiting vertex c with S. Equivalently, the mark M_i on S is not closed on S but it becomes closed on SS' for some sequence $S' \in \{0,1\}^*$.

We compute the desired upper bound in a graph-theoretic setting. Informally, for a marked traversal sequence S above, we construct an inconsistency graph $G_c(S)$ of S, whose vertex set represents all possible open marks on S and edge set represents all pairwise inconsistencies between open marks on S.

The algorithm for deciding if a particular S constitutes a leaf in the quadtree is summarized below.

algorithm Is_Leaf (S, c, ρ) *return boolean*
1. For each $i \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$, place closed and open marks of M_i on S;
2. For each non-empty prefix (pair-sequence) S' of S,
   2.1. Form the inconsistency graph $G_c(S')$ of open marks on S, and solve the Maximum Independent Set problem for $G_c(S')$;
   2.2. Compute the sum m of the number of closed marks on S' (in step 1) and the independence number of $G_c(S')$ (in step 2.1);
   2.3. If $m/\frac{|S'|}{2} \leq \rho$, then return *true*;
3. Return *false*;
end Is_Leaf

A successful termination of Build_Quadtree (c, $\frac{2^{c-1}}{r}$) gives the supposition in Lemma 1.

Theorem 2. *[8] If Build_Quadtree*(c, $\frac{2^{c-1}}{r}$) *successfully terminates for some positive integer* $c \geq 2$ *and positive real* $r \geq 2$*, then the supposition (hence the conclusion) is achieved.*

3 Refinements and Optimizations

In step 2 of the algorithm Is_Leaf, each upper-bound computation for the marks-to-pairs density of every non-empty prefix S' of segments S of a hypothetical reflecting sequence induces an instance of the Maximum Independent Set problem for $G_c(S')$. We can encompass all these instances with a unified approach: each $G_c(S')$ is an induced subgraph of the same inconsistency graph G_c, which captures all (pairwise) inconsistencies between all possible pairs of open marks on a traversal sequence.

For a positive integer c, the *inconsistency graph* G_c is defined as follows. The vertex set of G_c is $\{(C_\alpha, u) \mid \alpha \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$ and $u \in \{2, 3, \ldots, c-1\}$ is odd$\}$, in which a vertex (C_α, u) represents an open mark M_α on a non-empty traversal pair-sequence with the terminating vertex u (hence $1 < u < c$ and u is odd). The edge set of G_c represents pairwise inconsistencies between all possible pairs of $((C_\alpha, u), (C_\beta, v)) \in V(G_c) \times V(G_c)$ from one trivial and two non-trivial sources.

The trivial source of inconsistency results from that each traversal pair-sequence yields at most one open mark M_α for each $\alpha \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$, so we have $((C_\alpha, u), (C_\alpha, v)) \in E(G_c)$ for all $(C_\alpha, u), (C_\alpha, v) \in V(G_c)$ with $u \neq v$. The two non-trivial sources are suffix-inconsistency and prefix-inconsistency. Informally, the suffix-inconsistency reflects the co-existence of two open marks,

while the prefix-inconsistency represents the co-reachability of the open marks from their initial configuration.

For $\alpha \in \{0,1\}^*$, i.e., $\alpha = \alpha_1\alpha_2\cdots\alpha_n$ for some $n \geq 0$, where $\alpha_i \in \{0,1\}$ for $i = 0, 1, \ldots, n$, let $\alpha(i,j)$ denote $\alpha_i\alpha_{i+1}\cdots\alpha_j$ if $i \leq j$, and the empty sequence otherwise. We denote by $\overline{\alpha}$ the reversal of the component-wise complement of α, that is, $\overline{\alpha} = \overline{\alpha_n}\,\overline{\alpha_{n-1}}\cdots\overline{\alpha_1}$. To study the suffix-consistency of (C_α, u) and (C_β, v) for two labeled chains $C_\alpha, C_\beta \in \mathcal{L}(c)$ and two vertices $u, v \in \{1, 2, \ldots, c\}$ in C_α and C_β, respectively, we consider $\delta(u,v) = (c-u)-(v-1)$, which measures the difference of distances of u and v from vertices c and 1 in C_α and C_β, respectively.

The theorem below characterizes the suffix-inconsistency between (C_α, u), $(C_\beta, v) \in \mathcal{L}(c) \times \{2, 3, \ldots, c-1\}$ via $\delta(u,v)$ in the number of non-reflecting bits in $\alpha(2, c-1-\delta(u,v))$ versus $\beta(2, c-1-\delta(u,v))$. We define a first-order predicate $NR_{c,\alpha,\beta}$ for the maximum number of non-reflecting bits in such $\alpha, \beta \in \{0,1\}^{c-1}$ inductively as follows:

$$NR_{c,\alpha,\beta}(i) = \begin{cases} \beta(2, c-1-i) = \overline{\alpha(2, c-1-i)} & \text{if } i \text{ is either 0 or 1 ,} \\ (\beta(2, c-1-i) = \overline{\alpha(2, c-1-i)}) \vee & \\ \quad \exists j_i \in \{2, 3, \ldots, c-1-i\}(\beta_{j_i} = \alpha_{2+c-1-i-j_i} & \\ \quad \wedge NR_{c,\alpha,\beta}(i-2)) & \text{if } i \geq 2 \text{ .} \end{cases}$$

Theorem 3. *Let $(C_\alpha, u), (C_\beta, v) \in \mathcal{L}(c) \times \{2, 3, \ldots, c-1\}$, we have:*

1. *If $\delta(u,v) < 0$, then (C_α, u) and (C_β, v) are suffix-consistent.*
2. *If $\delta(u,v) \geq 0$, then (C_α, u) and (C_β, v) are suffix-inconsistent if and only if $NR_{c,\alpha,\beta}(\delta(u,v))$.*

3.1 Pruning Redundant Branches in the Quadtree

Every labeled chain $C \in \mathcal{L}(c)$ has an "ideal walk" starting at vertex 1 that represents the densest packing of reflections (marks) that C can exhibit; and that sequence of vertices visited depends only on c: $1, 2, \ldots, c, c-1, c-2, \ldots, 1, 0, 1, \ldots$. We can measure the goodness of every walk in a labeled chain C starting at vertex 1 by how much of the ideal walk in C starting at vertex 1 is covered.

We define the notion of weight formally as follows. Let $W(C,S)$ denote the walk induced by a traversal sequence $S \in \{0,1\}^*$ in a labeled chain $C \in \mathcal{L}(c)$ starting at vertex 1. A low reflection at vertex 0 represents a traversal from vertex c to vertex 0 with S; and a high reflection at vertex c is defined analogously. For a walk $W(C,S)$ terminating at vertex v, the ideal walk of $W(C,S)$, denoted by $\mathit{Ideal}(W(C,S))$, is the ideal walk in C starting and terminating at vertices 1 and v, respectively, which has the same number of reflections as $W(C,S)$. The *weight*, $wt(C,S)$, of a walk $W(C,S)$ is the length of the vertex-sequence of $\mathit{Ideal}(W(C,S))$.

The following two theorems justify the pruning of redundant branches in the quadtree.

Theorem 4. *For a labeled chain $C \in \mathcal{L}(c)$ and two traversal sequences S_1 and S_2 of the same length-parity, if $wt(C, S_1) \leq wt(C, S_2)$, then for every $S \in \{0,1\}^*$, $wt(C, S_1S) \leq wt(C, S_2S)$.*

Proof. Omitted. ∎

For $\alpha_1\alpha_2 \cdots \alpha_n \in \{0,1\}^n$, where
$n \geq 0$, (1) $\pi_1(\alpha_1\alpha_2\alpha_3\alpha_4 \cdots \alpha_n) = \alpha_1\overline{\alpha_2}\alpha_3\overline{\alpha_4} \cdots \alpha'_n$,
where $\overline{\alpha}$ denotes the complement of $\alpha \in \{0,1\}$, and α'_n is α_n if n is odd, and $\overline{\alpha_n}$ otherwise, (2) $\pi_2(\alpha_1\alpha_2\alpha_3\alpha_4 \cdots \alpha_n) = \overline{\alpha_1}\alpha_2\overline{\alpha_3}\alpha_4 \cdots \alpha''_n$, where α''_n is $\overline{\alpha_n}$ if n is odd, and α_n otherwise, and (3) $\pi_3 = \pi_2 \circ \pi_1$.

A more general form of redundancy is obtained via the functions π_1, π_2, and π_3. Let $1_{\{0,1\}^*}$ denote the identity function on $\{0,1\}^*$, and $\Pi = \{1_{\{0,1\}^*}, \pi_1, \pi_2, \pi_3\}$.

Theorem 5. *Let C_i be a labeled chain in $\mathcal{L}(c)$, where $i \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$, and $\pi \in \Pi$. For two traversal sequences S_1 and S_2 of even length-parity, if $wt(C_i, S_1) \leq wt(C_{\pi(i)}, S_2)$, then for every $S \in \{0,1\}^*$,*
$wt(C_i, S_1S) \leq wt(C_{\pi(i)}, S_2\pi(S))$.

Proof. We modify the induction argument in the proof of Theorem 4 as follows. For the induction step, suppose that the statement in the theorem is true for every $S \in \{0,1\}^*$ with $|S| \leq n$, where $n \geq 0$. Consider a sequence $S' \in \{0,1\}^{n+1}$, and assume that $wt(C_i, S_1) \leq wt(C_{\pi(i)}, S_2)$. Consider the two cases for the parity of n.

The case of even n: We write $S' = Sa$, where $S \in \{0,1\}^n$ and $a \in \{0,1\}$. Applying the induction hypothesis on S, we have $wt(C_i, S_1S) \leq wt(C_{\pi(i)}, S_2\pi(S))$. Note that $\pi(S') = \pi(Sa) = \pi(S)\pi(a)$.

If $wt(C_i, S_1S) = wt(C_{\pi(i)}, S_2\pi(S))$, then the terminating vertices of the two walks $W(C_i, S_1S)$ and $W(C_{\pi(S)}, S_2\pi(S))$ have the same vertex-label — with odd parity due to the even length-parity of S_1S and $S_2\pi(S)$. Starting at these two vertices with the same odd vertex-label in C_i and $C_{\pi(i)}$, the sequence of vertices visited in C_i induced by a traversal sequence $T \in \{0,1\}^*$ is identical to the sequence of vertices in $C_{\pi(i)}$ induced by the sequence $\pi(T)$. Thus the traversal sequences S_1Sa and $S_2\pi(S)\pi(a)$ in C_i and $C_{\pi(i)}$, respectively, satisfy that $wt(C_i, S_1Sa) = wt(C_{\pi(i)}, S_2\pi(S)\pi(a))$.

If $wt(C_i, S_1S) < wt(C_{\pi(i)}, S_2\pi(S))$, then we have
$wt(C_i, S_1S) + 2 \leq wt(C_{\pi(i)}, S_2\pi(S))$
as S_1 and S_2 are of the same length-parity. Notice that $|wt(C_i, S_1Sa) - wt(C_i, S_1S)| \leq 1$ and $|wt(C_{\pi(i)}, S_2\pi(S)\pi(a)) - wt(C_{\pi(i)}, S_2\pi(S))| \leq 1$, we have

$$wt(C_i, S_1Sa) \leq wt(C_i, S_1S)+1 \leq wt(C_{\pi(i)}, S_2\pi(S))-1 \leq wt(C_{\pi(i)}, S_2\pi(S)\pi(a)) .$$

The case of odd n: We write $S' = Sa_1a_2$, where $S \in \{0,1\}^{n-1}$ and $a_1, a_2 \in \{0,1\}$. Applying the induction hypothesis on S, we have $wt(C_i, S_1S) \leq wt(C_{\pi(i)}, S_2\pi(S))$. Notice that $\pi(S') = \pi(Sa_1a_2) = \pi(S)\pi(a_1a_2)$. We proceed as in the previous case, and the sub-case when assuming that $wt(C_i, S_1S) = wt(C_{\pi(i)}, S_2\pi(S))$ is similar.

We now assume that $wt(C_i, S_1S) < wt(C_{\pi(i)}, S_2\pi(S))$. As the two sequences S_1S and $S_2\pi(S)$ are of even length-parity, we have $wt(C_i, S_1S) + 2 \leq wt(C_{\pi(i)}, S_2\pi(S))$, and both terminating vertices of the walks $W(C_i, S_1S)$ and $W(C_{\pi(i)}, S_2\pi(S))$ ($W(C_i, S_1Sa_1)$ and $W(C_{\pi(i)}, S_2\pi(S)\pi(a_1))$) have odd (even, respectively) vertex-labels. Since $|wt(C, Ta) - wt(C, T)| \leq 1$ for all labeled chains $C \in \mathcal{L}(c)$, $T \in \{0,1\}^*$, and $a \in \{0,1\}$, we obtain that $wt(C_i, S_1Sa_1) \leq wt(C_{\pi(i)}, S_2\pi(S)\pi(a_1))$ and its strict inequality will give the desired result.

Consider now that $wt(C_i, S_1Sa_1) = wt(C_{\pi(i)}, S_2\pi(S)\pi(a_1))$. Both walks $W(C_i, S_1Sa_1)$ and $W(C_{\pi(i)}, S_2\pi(S)\pi(a_1))$ terminate at vertices with the same even vertex-label v in C_i and $C_{\pi(i)}$, respectively. If v is either 0 or c, we obtain the desired result. Let i_v and $\pi(i)_v$ denote the v^{th} bits of the labels i and $\pi(i)$ of C_i and $C_{\pi(i)}$, respectively — the "forward" edge-labels associated with vertex v in C_i and $C_{\pi(i)}$, respectively. Let a'_2 denote the last bit of $\pi(a_1a_2)$. We can verify that $i_v = a_2$ if and only if $\pi(i)_v = a'_2$. Thus both walks $W(C_i, S_1Sa_1a_2)$ and $W(C_{\pi(i)}, S_2\pi(S)\pi(a_1)a'_2)$ $(= W(C_{\pi(i)}, S_2\pi(S)\pi(a_1a_2)))$ terminate at vertices with the same vertex-label in C_i and $C_{\pi(i)}$, respectively. Hence $wt(C_i, S_1Sa_1a_2) = wt(C_{\pi(i)}, S_2\pi(S)\pi(a_1a_2))$.

Combining the two cases for the parity of n, we have $wt(C_i, S_1S') \leq wt(C_{\pi(i)}, S_2\pi(S'))$, as desired. This completes the induction step. By induction, the theorem is proved. ∎

For two traversal sequences S_1 and S_2 of even length-parity and $|S_1| \geq |S_2|$, if for some $\pi \in \Pi$, $wt(C_i, S_1) \leq wt(C_{\pi(i)}, S_2)$ for each $i \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$, then by Theorem 6, for every $S \in \{0,1\}^*$, $wt(C_i, S_1S) \leq wt(C_{\pi(i)}, S_2\pi(S))$ for each $i \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$. This gives that $\rho_1(S_1S) \leq \rho_1(S_2\pi(S))$ for every $S \in \{0,1\}^*$. Thus S_1 is a redundant branch to explore.

We define a binary *redundancy-ordering relation* $\leq_r$ on the set of even-length sequences of $\{0,1\}^*$ as follows. For $S_1, S_2 \in \{0,1\}^*$ of even length, $S_1 \leq_r S_2$ if and only if $|S_1| \geq |S_2|$, and for some $\pi \in \Pi$, $wt(C_i, S_1) \leq wt(C_{\pi(i)}, S_2)$ for each $i \in \{0^{c-1}, 0^{c-2}1, \ldots, 1^{c-1}\}$. Note that $\leq_r$ is a preorder (that is, a reflexive and transitive binary relation), and every pair of sequences of length 2 are related under $\leq_r$. We use the $\leq_r$-redundancy relation to prune the branches near the root of a quadtree, as well as locally at deeper depths.

3.2 The Shared Tree of Non-redundant Sequences

For a given chain length c and a pre-determined positive integer Δ, we generate a depth-Δ tree T_Δ, consisting of all non-redundant pair-sequences (that is, pair-sequences maximal respect to $\leq_r$) of length at most 2Δ, inductively level by level. Suppose that we have constructed the tree T_δ of all non-redundant pair-sequences of length at most 2δ, where $1 \leq \delta \leq \Delta$. Consider the set $L_{\delta+1}$ of all immediate descendant pair-sequences of the leaves of T_δ. Denote by $M(A)$ the set of all $\leq_r$-maximal elements in a set A of sequences of $\{0,1\}^*$. We compute the set $M(V(T_\delta) \cup L_{\delta+1})$ as follows:

1. Employ a simple divide-and-conquer algorithm to compute the set $M(L_{\delta+1})$, and

2. Compute the set $M(V(T_\delta) \cup L_{\delta+1})$ by examining all pairs in $V(T_\delta) \times M(L_{\delta+1})$ for $\leq_r$-redundancy.

The non-redundant sequences in the shared tree T_Δ are saved in a file and read each time the algorithm Build_Quadtree is using the chain length c. Before a pair-sequence of length at most 2Δ is examined, it is first looked up in the non-redundant sequence list; and if it is redundant, the sequence is not examined. At depth δ that exceeds the pre-determined Δ, we perform small-scale local redundancy analysis to the subtrees rooted at depth δ according to the $\leq_r$-redundancy relation.

3.3 Local Trees of Non-redundant Sequences

For a given chain length c and a pre-determined positive integer Δ', and a pair-sequence S of length $2(\Delta + i\Delta')$, where $i \geq 0$, we can cover S with the shared tree T_Δ and a sequence/stack of local trees $t_1, t_2, \ldots, t_i$, each of which is a depth-Δ' tree of locally non-redundant pair-sequences rooted at a prefix of S.

We can construct the sequence of local trees inductively on i in a manner analogous to the construction of the shared tree T_Δ described above. Before the local tree t_{i+1} is generated, the pair-sequence at its root is compared to the non-redundant sequence lists of $t_i, t_{i-1}, \ldots, t_1$; and if it is redundant, it is treated as a leaf. (In our current implementation, only the non-redundant sequence list of t_i is included for comparison.)

4 Conclusions

We have implemented the Quadtree Enumeration method with the improvements outlined in previous section (implementation is available upon request from the author), and have run the algorithm using a chain length of $c = 7$, a marks-to-pairs density upper bound of $\rho = \frac{64}{19}$, a $\leq_r$-redundancy shared tree of depth $\Delta = 10$, and local trees of depth $\Delta' = 4$. This has resulted in $R(t, 7) \geq 19t - 214$ and the length lower bounds for universal traversal sequences as in [8] with significant speedup, excluding the computation time of the $\leq_r$-redundancy shared tree.

Our current effort is to employ the Quadtree Enumeration method incorporating the above-mentioned refinements and optimizations for a larger chain length of 9. Future work is directed at finding fast algorithms for solving the embedded Maximum Independent Set problems for induced subgraphs of inconsistency graphs, and at improving the construction of the $\leq_r$-redundancy table.

References

1. R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the 20th IEEE Symposium on the Foundations of Computer Science*, pages 218–223. IEEE Computer Society, October 1979.

2. P. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa. Time-space tradeoffs for undirected graph traversal. In *Proceedings of the 31st IEEE Symposium on the Foundations of Computer Science*, pages 429–438. IEEE Computer Society, October 1990.
3. A. Bar-Noy, A. Borodin, M. Karchmer, N. Linial, and M. Werman. Bounds on universal sequences. *SIAM Journal on Computing*, 18(2):268–277, 1989.
4. A. Borodin, W. L. Ruzzo, and M. Tompa. Lower bounds on the length of universal traversal sequences. In *Proceedings of the Twenty-First ACM Symposium on the Theory of Computing*, pages 562–573. Association for Computing Machinery, May 1989.
5. J. Buss and M. Tompa. Lower bounds on universal traversal sequences based on chains of length five. *Information and Computation*, 120(2):326–329, 1995.
6. S. A. Cook and C. W. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal on Computing*, 9(3):636–652, 1980.
7. A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover times. In *Proceedings of the Twenty-First ACM Symposium on the Theory of Computing*, pages 574–586. Association for Computing Machinery, May 1989.
8. H. K. Dai and K. E. Flannery. Improved length lower bounds for reflecting sequences. In J.-Y. Cai and C. K. Wong, editors, *Lecture Notes in Computer Science (1090): Computing and Combinatorics*, pages 56–67, Springer-Verlag, Berlin Heidelberg, 1996.
9. M. Tompa. Lower bounds on universal traversal sequences for cycles and other low degree graphs. *SIAM Journal on Computing*, 21(6):1153–1160, 1992.

Competitive Facility Location along a Highway*

Hee-Kap Ahn[1], Siu-Wing Cheng[2], Otfried Cheong[1],
Mordecai Golin[2], and René van Oostrum[1]

[1] Department of Computer Science, Utrecht University, Netherlands,
{heekap,otfried,rene}@cs.uu.nl
[2] Department of Computer Science, HKUST, Hong Kong,
{scheng,golin}@cs.ust.hk

Abstract. We consider a competitive facility location problem with two players. Players alternate placing points, one at a time, into the playing arena, until each of them has placed n points. The arena is then subdivided according to the nearest-neighbor rule, and the player whose points control the larger area wins. We present a winning strategy for the second player, where the arena is a circle or a line segment.

1 Introduction

The classical facility location problem [5] asks for the optimum location of a new facility (police station, super market, transmitter, etc.) with respect to a given set of customers. Typically, the function to be optimized is the maximum distance from customers to the facility — this results in the minimum enclosing disk problem studied by Megiddo [8], Welzl [12] and Aronov et al. [2].

Competitive facility location deals with the placement of sites by competing market players. Geometric arguments are combined with arguments from *game theory* to see how the behavior of these decision makers affect each other. Competitive location models have been studied in many different fields, such as spatial economics and industrial organization [1,9], mathematics [6] and operations research [3,7,11]. Comprehensive overviews of competitive facility locations models are the surveys by Friesz et al. [11], Eiselt and Laporte [3] and Eiselt et al. [4].

We consider a model where the behavior of the customers is deterministic in the sense that a facility can determine the set of customers more attracted to it than to any other facility. This set is called the *market area* of the facility. The collection of market areas forms a tessellation of the underlying space. If customers choose the facility on the basis of distance in some metric, the tessellation is the Voronoi Diagram of the set of facilities [10].

We address a competitive facility location problem that we call the *Voronoi Game*. It is played by two players, Blue and Red, who place a specified number, n,

* Part of the work was done while the first, third, and fifth authors were at the Dept. of Computer Science, HKUST, Hong Kong. The work described in this paper has been supported by the Research Grants Council of Hong Kong, China (HKUST6074/97E, HKUST8088/99E, HKUST6094/99E, HKUST6162/00E, and HKUST6137/98E).

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 237–246, 2001.
© Springer-Verlag Berlin Heidelberg 2001

of facilities in a region U. They alternate placing their facilities one at a time, with Blue going first. After all $2n$ facilities have been placed, their decisions are evaluated by considering the *Voronoi diagram* of the $2n$ points. The player whose facilities control the larger area wins.

More formally, let $\{b_i\}_{i=1}^n$ and $\{r_i\}_{i=1}^n$ be the respective locations of the blue and red points and set

$$\mathcal{B} = |\{u \in U \,:\, \min_i d(u, b_i) < \min_i d(u, r_i)\}|,$$
$$\mathcal{R} = |\{u \in U \,:\, \min_i d(u, r_i) < \min_i d(u, b_i)\}|$$

where $d(u, v)$ is an underlying metric and $|\cdot|$ indicates the area of a set. Blue wins if and only if $\mathcal{B} > \mathcal{R}$, Red wins if and only if $\mathcal{R} > \mathcal{B}$ and the game ends in a tie if $\mathcal{B} = \mathcal{R}$.

The most natural Voronoi Game is played in a two-dimensional arena U using the Euclidean metric. Unfortunately nobody knows how to win this game, even for very restricted regions U. In this note we present strategies for winning one-dimensional versions of the game, where the arena is a circle or a line segment, and variations. In other words, we consider competitive facility location along an Australian highway.

The next section discusses the simplest game, on the circle. It is obvious that the second player, Red, can always achieve a tie by playing on the antipode of Blue's move. One might try to tweak this strategy such that it results in a win for Red. This doesn't seem to work, and we present instead a quite different winning strategy for Red.

Section 3 describes how this strategy remains a winning strategy even if the rules of the game are drastically relaxed.

In Section 4 we finally turn to the line segment arena. It would appear that Blue has an advantage here, because it can play the midpoint of the segment in its first move. We show that this doesn't help, and prove that Red still has a winning strategy. The strategy is quite similar to the one for the circle case, but its analysis (because of a loss of symmetry) is more detailed.

2 The Basic Circle Game

There are two players, Blue and Red, each having n *points* to play, where $n > 1$. They alternate placing these points on circle C, with Blue placing the first point, Red the second, Blue the third, etc., until all $2n$ points are played. We assume that points cannot lie upon each other. Let $\{b_i\}_{i=1}^n$ be the locations of the blue points and $\{r_i\}_{i=1}^n$ be those of the red ones. After all of the $2n$ points have been played each player receives a score equal to the total circumference of the circle that is closer to that player than to the other, i.e., Blue and Red have respective scores

$$\mathcal{B} = |\{x \in C \,:\, \min_i d(x, b_i) < \min_i d(x, r_i)\}|$$
$$\mathcal{R} = |\{x \in C \,:\, \min_i d(x, r_i) < \min_i d(x, b_i)\}|$$

The player with the highest score (the larger circumference) wins.

The question that we address here is, *Does either player have a winning strategy and, if yes, what is it?* We will see below that the second player, Red, always has a winning strategy.

Before giving the strategy we introduce some definitions. We parameterize the circle using the interval $[0, 1]$, where the points 0 and 1 are identified. Arcs on the circle are written as $[x, y]$ implying the clockwise arc running from x to y, as in $[.5, .6]$ or $[.9, .1]$.

Definition 1. *The n points $u_i = \frac{i}{n}$, $i = 0, 1, \ldots, n-1$ are* **keypoints**.

We call an arc between two clockwise consecutive red/blue points an *interval*. The interior of an interval is free of red/blue points. At any given time during the game the circle is partitioned into intervals. An interval is *monochromatic* if its endpoints have the same color, and *bichromatic* if they have different colors. A *blue interval* is a blue monochromatic one, a *red interval* a red monochromatic one. We denote the total length of all red intervals by R_m, and the total length of all the blue intervals by B_m. An interval is called a *key interval* if both of its endpoints are keypoints.

The important thing to notice is that at the end of the game the length of each bichromatic interval is divided equally among the two players, so $\mathcal{R} - \mathcal{B} = R_m - B_m$ and Red wins if and only if $R_m > B_m$. We devise our strategy to force this to happen.

Since we can parameterize the circle arbitrarily, we can assume without loss of generality that Blue plays his first point on 0 and thus on a keypoint. We now describe Red's winning strategy. Figure 1 shows an example.

Red's Keypoint Strategy

Stage I: If there is an empty keypoint then Red plays onto the keypoint.
Stage I ends after the last keypoint is played (by either Red or Blue).

Stage II: If there is no empty keypoint and it is not Red's last move then Red plays her point into a largest blue interval. We call this *breaking* the blue interval.
Stage II ends when Blue plays his last point.

Stage III: Red's last move. There are two possibilities:

(i) if there exists more than one blue interval then Red breaks a largest one by placing her point inside.

(ii) if there is only one blue interval define $\ell < \frac{1}{n}$ to be its length. Red's move is to go to a bichromatic key interval and claim a red interval of length larger than ℓ by placing a red point closer than $\frac{1}{n} - \ell$ to the blue endpoint of the bichromatic key interval.

The two following lemmas will be needed.

Lemma 1. *Let B be a set of b blue points and let R be a set of r red points currently on the circle with $b \geq r$. Let $n(R)$ be the number of red intervals they form and $n(B)$ the number of blue ones. Then $n(B) - n(R) = b - r$.*

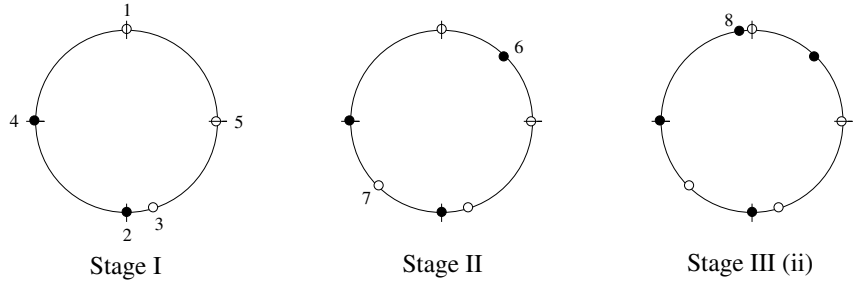


Fig. 1. There are four points to be played for both Blue and Red. The white dots represent Blue's points and the black dots represent Red's points. We label the dots in chronological order.

Proof. The proof will be by induction on r. If $r = 0$ then b blue points form b blue intervals so $n(B) = b$, $n(R) = 0$ and the condition $n(B) - n(R) = b - r$ is satisfied.

Now suppose that the lemma is true for all configurations of b blue points and $r - 1$ red ones. Deleting any red point p from R leaves b blue and $r - 1$ red points, so $n(B) - n(R - \{p\}) = b - r + 1$. We now add the red point p back into the configuration and ask how the monochromatic intervals can change. There are three possible placements of p :

(i) inside a red interval, increasing $n(R)$ by one and leaving $n(B)$ unchanged.
(ii) inside a blue interval, decreasing $n(B)$ by one and leaving $n(R)$ unchanged.
(iii) inside a bichromatic interval, increasing $n(R)$ by one and leaving $n(B)$ unchanged.

After all three of these cases we find that $n(B) - n(R) = n(B) - n(R - \{p\}) - 1 = b - r$.

Lemma 2. *Suppose that all n keypoints are covered and Blue has just moved (possibly covering the last keypoint). If there is only one blue interval and this interval has length $< \frac{1}{n}$, then there exists a bichromatic key interval.*

Proof. We apply the pigeon hole principle: At most $2n - 1$ points have been played, n of them on keypoints. Consider the n circle arcs of length $\frac{1}{n}$ formed by the n keypoints. Since the blue interval has length $< \frac{1}{n}$, at least one of its endpoints is *inside* an arc. That leaves only $n - 2$ points to have been played inside the $n - 1$ remaining arcs. Therefore, one of the arcs must be free of points, forming a key interval. Since there is only one blue interval, there is no red interval by Lemma 1. Therefore, this key interval is bichromatic.

Theorem 1. *The keypoint strategy is a well-defined winning strategy for Red.*

Proof. We start with a simple observation. Since the circle contains only n keypoints and Blue's first move covers the first keypoint, Red will play onto at most $n - 1$ keypoints. Thus Stage I always ends before Red plays her last point.

Consider Stage II. Lemma 1 implies that after each play by Blue ($b = r + 1$) there is always at least one blue interval on the circle, so Stage II of the strategy is indeed well defined.

We make two observations concerning the situation after Stage II, when Red has played her $n - 1$'st point. The first is that *there is no blue key interval*. Let k be the number of keypoints played by Blue during the game. Red has covered the remaining $n - k$ keypoints by the end of Stage I. If $k = 1$ (the only case in which Red skips Stage II), then there certainly is no blue key interval as there is only one blue keypoint. When $k > 1$, Blue can define at most $k - 1$ blue key intervals with its k keypoints (since Red has at least one keypoint). Note that since all keypoints are played by the end of Stage I, all intervals in Stages II and III have length at most $\frac{1}{n}$. In particular a blue key interval is longer than any other blue interval. Since Red plays $k - 1$ points in Stage II, all blue key intervals are broken during Stage II.

The second observation concerning the situation after stage II is that *all red intervals are key intervals*. This statement is true at the end of Stage I, as Red has so far only played onto keypoints, and all keypoints are covered. During Stage II, Red uses her points to break blue intervals, and therefore creates bichromatic intervals only. Blue cannot create red intervals, and so, at the end of Stage II, all red intervals are indeed key intervals.

We now show why Stage III is well defined and why Red wins. Suppose that Blue has just played his last point and it is now time for Stage III, Red's last move. From Lemma 1 we know that $n(B) \geq 1$.

If $n(B) > 1$ before Red's last move then the strategy is well defined: Red breaks a largest blue interval. This decreases $n(B)$ by 1 so the game ends with $n(B) \geq 1$. By Lemma 1 we have $n(R) = n(B) \geq 1$. But now note that from the observations in the preceding paragraphs all existing red intervals are key intervals while all existing blue intervals have length strictly less than $\frac{1}{n}$. Since all red intervals are longer than all blue intervals and there are the same number of red ones and blue ones we find that $R_m > B_m$ and Red wins.

If $n(B) = 1$ before Red's last move the strategy requires that the unique blue interval has length $\ell < \frac{1}{n}$, and that there exists a bichromatic key interval. The first fact was already observed above, the second fact follows from Lemma 2.

After Red places her last point Blue still has one blue interval of length ℓ while Red has one red interval of length $> \ell$. Thus $R_m > \ell = B_m$ and Red wins.

3 A Modified Circle Game

The basic game can be modified in many different ways. The simplest modification allows the players to play more than one point at a time. More complicated modifications permit the players (both or one) to choose before each turn, how many points they play.

Suppose that there are $k \leq n$ rounds. Let β_i and γ_i be the numbers of points that Blue and Red play respectively in round i. Suppose that the following restrictions are placed.

- $\forall 1 \leq i \leq k,\ \beta_i, \gamma_i > 0$.
- $\forall 1 \leq j \leq k,\ \sum_{i=1}^{j} \beta_i \geq \sum_{i=1}^{j} \gamma_i$.
- $\sum_{i=1}^{k} \beta_i = \sum_{i=1}^{k} \gamma_i = n$.
- $\beta_1 < n$.

Then Red still wins by following exactly the same strategy as in the previous section of first filling in the keypoints and then breaking the largest blue intervals until Red plays it's last point when it follows the Stage III rules. The proof that the strategy is well defined and wins is almost exactly the same as the one in the previous section so we will not repeat it here.

Note that this generalization includes both the original game and the "batched" version in which each player plays the same number (> 1) of points at each turn. Note, too, that k, β_i, and γ_i need not be fixed in advance. For example, Blue may decide at every move how many points he will play and then Red plays the same number.

We conclude this section by noting that the condition $\beta_1 < n$ is essential since otherwise Blue would play the keypoints, forcing a tie.

4 The Line Segment Version

We now move on to the version of the game played on a line segment. We consider it to be horizontal and parameterized as $[0, 1]$. The scoring is the same as in the basic circle game except that the player with the leftmost point claims everything between 0 and the point, and the player with the rightmost point claims everything between the point and 1. We assume that $n > 1$, and points cannot lie upon each other. When $n = 1$, Blue wins by placing onto $\frac{1}{2}$.

We modify some of the old definitions and introduce new ones:

Definition 2. *The n points $u_i = \frac{1}{2n} + \frac{i}{n}$, $i = 0, 1, \ldots, n-1$ are* **keypoints**.

The *left segment* is the segment from 0 to the leftmost red or blue point. The *right segment* is the segment from the rightmost red or blue point to 1. The *border interval* is the union of the left and right segments. An *interval* is a section of the line segment with red/blue endpoints and no red/blue points in its interior. We consider the *border interval* an *interval*. An interval, including the border interval, is *monochromatic* if its endpoints have the same color, and *bichromatic* if they have different colors. With this definition of intervals, Lemmas 1 and 2 are true for the line segment as well.

We denote the total length of all of the blue intervals—including, if appropriate, the border interval—by B_m, the total length of all of the red intervals—again including, if appropriate, the border interval—by R_m. When the border interval is bichromatic, we use B_b to denote the length of the left/right segment with a blue endpoint and R_b to denote the length of the left/right segment with a red endpoint. If the border interval is monochromatic, then $B_b = R_b = 0$. Since all bichromatic non-border intervals are equally shared by both players $\mathcal{R} - \mathcal{B} = (R_m + R_b) - (B_m + B_b)$ and, as in Section 2, we design our strategy so that Red finishes with the right hand side of the equation > 0.

We now introduce the *line strategy*, a modified version of the circle strategy. Figure 2 shows an example.

Red's Line strategy

Stage I: If there is an empty keypoint then Red plays the keypoint. If u_0 or u_{n-1} have not yet been played then Red should play onto one of them first. Stage I ends after the last keypoint is played by either Red or Blue. Note that the game may finish in Stage I.

Stage II: If there is no empty keypoint and it is not Red's last move then

(i) if there exists at least one blue non-border interval, then Red should break a largest blue non-border interval by placing her point inside.

(ii) if the border interval is the only blue interval, then there are two possible cases:

(a) One of the blue endpoints of the blue border interval is a keypoint:
Without loss of generality assume that it is u_0 (the other case is symmetric) and the other endpoint is $1 - \ell$. From Stage I the other endpoint cannot be the keypoint u_{n-1} so $\ell < \frac{1}{2n}$. Red now places her new point at x where x is anywhere in (ℓ, u_0).

(b) Neither of the endpoints of the border interval are keypoints: Let ℓ be the length of the blue border interval; $\ell < \frac{1}{n}$. There must then exist a bichromatic key interval (Lemma 2). Red places her new point in that interval to form a new red interval of length $> \ell$.

Stage II ends after Blue plays his last point.

Stage III: If Red is placing her last point, we have two mutually exclusive cases:

(i) if there exists more than one blue interval, then Red should break a largest non-border one.

(ii) if there exists only one blue interval, then let its length be ℓ; we will see below that $\ell < \frac{1}{n}$. Red should go to a bichromatic key interval (one will exist from Lemma 2) and claim a red interval of length $> \ell$ as follows.

- If the bichromatic key interval is not the border one, Red can do this by creating a new red interval of length $> \ell$.
- If the bichromatic key interval is the border one, then Red already possesses $\frac{1}{2n}$ of it because it has all of either $[0, u_0]$ or $[u_{n-1}, 1]$. Red can therefore go to the other side it does not possess, and grab enough length to have a red border interval of length $> \ell$.

Theorem 2. *The line strategy is a well-defined winning strategy for Red.*

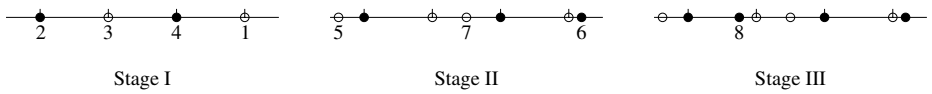


Fig. 2. There are four points to be played for both Blue and Red. The white dots represent Blue's points and the black dots represent Red's points. We label the dots in chronological order.

Proof. Note that this strategy differs in at least one major aspect from the circle strategy: since we have lost circular symmetry it cannot be guaranteed that Blue plays onto at least one keypoint, and so it is possible that the game will end in Stage I, with Red playing all n keypoints. In this case, all red intervals (including, possibly, the border interval) are key intervals and all blue intervals have length $< \frac{1}{n}$. By Lemma 1 Blue and Red have the same number of monochromatic intervals, so $B_m < R_m$. If the border interval is monochromatic, then $B_b = R_b = 0$ and Red wins. If the border interval is bichromatic, then one of its endpoints must be the red point u_0 or u_{n-1}. This implies that $B_b < R_b = \frac{1}{2n}$, and Red wins.

In what follows we may therefore assume that Blue plays onto at least one keypoint during the game. We will show that at the end of the game it will always be true that $B_m < R_m$ and, if the border interval is bichromatic, then $B_b \leq R_b$. The theorem will follow.

First note that under this assumption Stage I always ends with all keypoints covered, and Stage III is reached. Note further that Red's first move is onto either u_0 or u_{n-1}.

We consider Stage II. After Blue's every move there exists at least one blue interval (possibly the border interval) by Lemma 1. If there is only one such blue interval, there is no red interval. Thus, one of the two conditions (i) or (ii) of Stage II holds. The strategy is clearly well defined in cases (i) and (ii)(a), the validity of (ii)(b) follows from Lemma 2.

We will need one more observation.

Lemma 3. *After Blue's last move, there is no blue key interval.*

Proof. Let k be the number of keypoints played by Blue, where $1 \leq k \leq \lceil \frac{n}{2} \rceil$. There are therefore at most $k-1$ blue key intervals after Stage I. Red occupies $n-k$ keypoints in Stage I, and so Stage II lasts for $k-1$ rounds. This is sufficient for all blue key intervals to be broken, since a blue key interval is longer than any other blue interval in Stage II (i) and Stage II (ii)(a).

We now prove that Red wins. There are two cases.

Lemma 4. *Assume that case (ii) of Stage II never occurs. Then all red intervals are key intervals after Blue's last move.*

Proof. After Stage I all red intervals are key intervals, since Red has only played keypoints and all keypoints are covered. During Stage II

Red uses all her points to break blue intervals (since case (ii)(b) does not occur), and so creates only bichromatic intervals. As Blue cannot create red intervals, all red intervals remaining after Stage II are indeed key intervals.

We examine the result of Stage III, assuming that case (ii) of Stage II did not occur. Assume first that Red plays case (i) of Stage III. There are then equal numbers of red and blue intervals left after the last move. Since all blue intervals have length $< \frac{1}{n}$ by Lemma 3, and all red intervals are key intervals by Lemma 4, we have $B_m < R_m$. If the border interval is monochromatic, then $B_b = R_b = 0$ and Red wins. If the border interval is bichromatic, then its red endpoint must be u_0 or u_{n-1} (since Red plays case (i) of Stage II only), and so $B_b \leq R_b = \frac{1}{2n}$ and Red wins.

On the other hand, assume now that Red plays case (ii) of Stage III. Then Blue has total length ℓ, Red has $> \ell$, and so Red wins.

We now consider the remaining case, where case (ii) of Stage II *does* occur.

Lemma 5. *Assume that case (ii) of Stage II occurs at least once. After Red's last move in Stage II, there is no blue non-border interval and* $\mathcal{B} < \mathcal{R}$.

Proof. We prove that the statement is true after the *last* occurrence of case (ii) in Stage II, and after each subsequent move by Red.
Consider the last occurrence of case (ii). Before Red's move, the border interval is the only blue interval.
If Red plays case (ii)(a), there is no blue interval at all after Red's move. We have then $B_m = R_m = 0$ and $B_b < R_b$, which implies $\mathcal{B} < \mathcal{R}$.
If Red plays case (ii)(b), Red claims a red interval longer than the blue border interval. So after Red's move we have $\mathcal{B} < \mathcal{R}$, as $B_m < R_m$ and $B_b = R_b = 0$.
Consider now the remaining moves of Stage II. In all these moves Red plays case (i), and so we can deduce that Blue uses his move to create a new blue non-border interval. Red immediately breaks this blue interval. This leaves B_m, R_m, B_b, and R_b unchanged and destroys the only blue non-border interval. The claim therefore remains true after each subsequent move by Red, and in particular after Red's last move in Stage II.

We consider the situation right before Blue's last move. By Lemma 5, there is no blue non-border interval, and $\mathcal{B} < \mathcal{R}$.

If the border interval is blue before Blue's last move and Blue uses that move to create a blue non-border interval move, Red plays case (i) of Stage III. This breaks the new blue non-border interval, returning $\mathcal{B}$ and $\mathcal{R}$ to their state before Blue's last move, and so Red wins.

In all other cases there is a single blue interval after Blue's last move. It cannot be a key interval by Lemma 3, and so, by Lemma 2, there is a bichromatic key interval. Red claims a red interval longer than the blue interval, and wins.

5 Conclusions

We have given strategies for one-dimensional competitive facility location, allowing the second player, Red, to win. The margin by which Red wins is very small, however, and in fact Blue can make it as small as he wants. Moreover, our arguments can easily be transformed into a strategy that allows the first player, Blue, to lose by an arbitrarily small margin. Do our findings have any bearing on the two-dimensional Voronoi Game? The concept of *keypoints* turned out to be essential to our strategies. Is there a similar concept in two dimensions? A natural attempt would be to define a uniform square grid of keypoints. Perhaps surprisingly, a player governing this grid can still lose the game by a considerable margin.

Acknowledgments

We thank Dr. Jacob Ecco for introducing us to this problem and we also thank Sunil Arya, Mark de Berg, and Joachim Gudmundsson for helpful discussions.

References

1. S.P. Anderson. Equilibrium existence in a linear model of spatial competition. *Economica*, 55:479–491, 1988.
2. B. Aronov, M. van Kreveld, R. van Oostrum, and K. Varadarajan. Facility location on terrains. In K.Y. Chwa and Oscar H. Ibarra, editors, *Proc. 9th International Symposium of Algorithms and Computation*, volume 1533 of *Lecture Notes Comput. Sci.*, pages 19–28. Springer-Verlag, 1998.
3. H.A. Eiselt and G. Laporte. Competitive spatial models. *European Journal of Operational Research*, 39:231–242, 1989.
4. H.A. Eiselt, G. Laporte, and J.-F. Thisse. Competitive location models: A framework and bibliography. *Transportation Science*, 27:44–54, 1993.
5. R.L. Francis and J.A. White. *Facility Layout and Location.* Prentice-Hall, Inc., 1974.
6. S.L. Hakimi. Location with spatial interactions: Competitive location and games. In R.L. Francis and P.B. Mirchandani, editors, *Discrete Location Theory*, pages 439–478. Wiley, New York, 1990.
7. M. Labbé and S.L. Hakimi. Market and locational equilibrium for two competitors. *Operations Research*, 39:749–756, 1991.
8. N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput.*, 12:759–776, 1983.
9. A. Okabe and M. Aoyagy. Existence of equilibrium configurations of competitive firms on an infinite two-dimensional space. *Journal of Urban Economics*, 29:349–370, 1991.
10. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, NY, 1985.
11. R.L. Tobin T.L. Friesz and T. Miller. Existence theory for spatially competitive network facility location models. *Annals of Operations Research*, 18:267–276, 1989.
12. Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lecture Notes Comput. Sci.*, pages 359–370. Springer-Verlag, 1991.

Membership for Core of LP Games and Other Games*

Qizhi Fang[1], Shanfeng Zhu[2], Maocheng Cai[3], and Xiaotie Deng[2]

[1] Department of Applied Mathematics, Qingdao Ocean University, Qingdao 266003, P. R. China
[2] Department of Computer Science, City University of Hong Kong, Hong Kong, P. R. China
[3] Institute of Systems Science, Chinese Academy of Sciences, Beijing 100080, P. R. China

Abstract. Let $\Gamma \equiv (N, v)$ be a cooperative game with the player set N and characteristic function $v : 2^N \to R$. An imputation of the game is in the core if no subset of players could gain advantage by splitting from the grand coalition of all players. It is well known that, for the linear production game, and the flow game, the core is always non-empty (and a solution in the core can be found in polynomial time). In this paper, we show that, given an imputation x, it is *NP*-complete to decide it is not a member of the core, in both games. The same also holds for Steiner tree game. In addition, for Steiner tree games, we prove that testing the total balacedness is *NP*-hard.
Key words: cooperative game, core, network flow, linear programming, Steiner tree, *NP*-completeness.

1 Introduction

A cooperative game $\Gamma = (N, v)$ consists of a player set $N = \{1, 2, \cdots, n\}$ and a characteristic function $v : 2^N \to R$. For a revenue game Γ, the core is defined by

$$\text{Core}(\Gamma) = \{x \in R^n : x(N) = v(N) \text{ and } x(S) \geq v(S),\ \forall S \subseteq N\},$$

where $x(S) = \sum_{i \in S} x_i$. The study of the core are closely associated with another important concept, the balanced set. The collection $\mathcal{B}$ of subsets S of N is balanced if there exists a set of positive numbers γ_S, $S \in \mathcal{B}$, such that for each $i \in N$, we have $\sum_{i \in S \in \mathcal{B}} \gamma_S = 1$. A game (N, v) is called balanced if $\sum_{S \in \mathcal{B}} \gamma_S v(S) \leq v(N)$ holds for every balanced collection $\mathcal{B}$ with weights $\{\gamma_S : S \in \mathcal{B}\}$. With techniques essentially the same as the linear programming duality, Shapley [13] proved that a game has a non-empty core if and only if it is balanced. For a subset $S \subseteq N$, we define the induced subgame (S, v_S) on S: $v_S(T) = v(T)$ for

* Research is partially supported by a grant from the Research Grants Council of Hong Kong SAR (CityU 1116/99E) and a grant from CityU of Hong Kong (Project No. 7001215).

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 247–256, 2001.
© Springer-Verlag Berlin Heidelberg 2001

every subset T of S. A game is called totally balanced if all its subgames are balanced, i.e., all its subgames have non-empty cores.

The computational complexity as a rationality measure for game theoretical concepts has attracted more and more attention recently. In cooperative game studies, work has focused on games that have a succinct representation, that is, the game value is defined as a solution to a combinatorial optimization probelm, for which the input size is polynomial in the number of players, since first suggested by Megiddo [11]. Various interesting complexity structures have started to emerge as a result, especially for the study of core, which will be the focus of our paper. Deng and Papadimitriou [4] found a game for which the core is nonempty if and only if a certain imputation (Shapley value in this case) is in the core. Faigle, et al. [5] show that, for the minimum cost spanning tree (MCST) game, membership testing is *NP*-complete. It is known, however, the MCST game has a non-empty core and a member in the core can be found in polynomial time [8]. Goemans and Skutella [7] recently showed that, for a facility location game, if the core is non-empty, a solution in core can be found in polynomial time, and membership testing can be done in polynomial time. However, it is *NP*-complete to decide if a core is not empty. In this paper, we consider the flow game and the linear production game. Both are known to have a non-empty core and a solution in the core can be found in polynomial time [9,10,12,2]. However, we show that, given a solution, it is *NP*-complete to tell it is not in the core.

The flow game was introduced by Kalai and Zemel [9,10]. It is associated with the maximum flow from a source to a sink on a network, where each player controls one arc in the network. The value of a subset of players is the same as the maximum flow from the source to the sink on the subgraph induced by the edges controlled by that subset of players. The flow game is totally balanced, and on the other hand, every non-negative totally balanced game can be formulated as a flow game. Owen [12] introduced a linear production game in which each player j controls a certain resource vector b^j. Jointly, they maximize a linear objective function cx, subject to the resource constraint $Ax \leq \sum_{all\ j} b^j$. The value a subset S of players can achieve on their own is the maximum they can achieve with resources collectively owned by this subset: $\max\{cx : Ax \leq \sum_{j \in S} b^j\}$. Owen showed that the core for linear production game is always nonempty and one imputation in the core can be immediately obtained from his proof using variables of the dual linear program. Curiel [1] proved that the class of linear programming games is also equivalent to the class of totally balanced games. Therefore, the class of linear programming games is equivalent to the class of flow games. These reductions for the equivalence proof, however, involve in exponential time and space in the number of players. Therefore, computational complexity results for these two games may not be equivalent.

In Section 2, we first show that membership test in the core is *co-NP*-complete for flow games. Because of the special structure we have for the reduction, the result flow game has a polynomial size linear production game formulation. Therefore, the same conclusion for linear production games follows as a result. In Sec-

tion 3, we consider the same problem for the Steiner tree game and show it is also NP-hard.

In Section 4, we show it is NP-hard to decide whether a Steiner tree game is total balanced. Naturally, linear production game and flow game are total balanced. Deng, et al., considered total balanceness of several interesting combinatorial optimization games [3]. Most were shown to be polynomially decidable. The worst complexity is for a coloring game, for which total balancedness is equivalent to a graph being perfect graph. Our result is the first example of NP-hardness for total balanced condition.

2 Complexity of Membership Test in the Core for Flow Games and Linear Production Games

Theorem 2.1 It is NP-complete to show that, given a maximum flow game $\Gamma_D = (E, v)$ on network $D = (V, E; \omega)$ and a vector $x \in R^n$ $(n = |E|)$ with $x(E) = v(E)$, is x not a core member of the game, i.e., does there exist a coalition $S \subset E$ such that $x(S) < v(S)$?

Proof. The problem is in NP because a coalition $S \subset E$ with $x(S) < v(S)$ (if it exists) can be exhibited and checked easily. To prove completeness, we establish a polynomial transformation from a basic NP-complete problem, EXACT COVER BY 3-SETS [6]:

Given a finite set $X = \{x_1, x_2, \cdots, x_{3q}\}$ and a collection $F = \{f_1, f_2, \cdots, f_{|F|}\}$ of 3-element subsets of X $(|F| \geq q)$, does F contain an exact cover for X, that is, does there exist a subcollection $F' \subseteq F$ such that every element of X occurs in exactly one member of F' ?

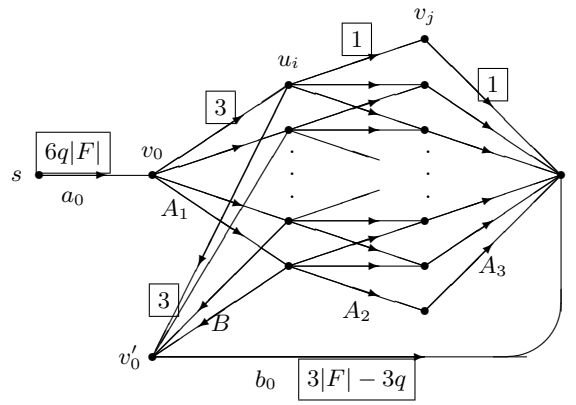
For an instance of X3C, we construct a maximum flow game Γ_D on network $D = (V, E; \omega)$:

Thus, the vertex set of D is $V = V_X \cup V_F \cup \{s, t, v_0, v_0'\}$. (See Fig.1).

- $V_X = \{v_1, v_2, \cdots, v_{3q}\}$, one element-vertex $v_i \in V_X$ for each $x_i \in X$, $i = 1, 2, \cdots, 3q$;
- $V_F = \{u_1, u_2, \cdots, u_{|F|}\}$, one set-vertex $u_j \in V_F$ for each $f_j \in F$, $j = 1, 2, \cdots, |F|$;
- source s, sink t, two additional vertices v_0 and v_0'.

The arc set E and the corresponding capacity function $\omega : E \to R^+$ are given as follows:

- $a_0 = (s, v_0), \quad \omega(a_0) = 6q|F|$;
- $b_0 = (v_0', t), \quad \omega(b_0) = 3|F| - 3q$;
- $A_1 = \{(v_0, u_i) : i = 1, 2, \cdots, |F|\}, \quad \forall e \in A_1 : \omega(e) = 3$;
- $A_2 = \{(u_i, v_j) : x_j \in f_i, i = 1, 2, \cdots, |F|, j = 1, 2, \cdots, 3q\}, \quad \forall e \in A_2 : \omega(e) = 1$;
- $A_3 = \{(v_j, t) : j = 1, 2, \cdots, 3q\}, \quad \forall e \in A_3 : \omega(e) = 1$;
- $B = \{(u_i, v_0') : i = 1, 2, \cdots, |F|\}, \quad \forall e \in B : \omega(e) = 3$.



$(i = 1, 2, \cdots, |F|;\ j = 1, 2, \cdots, 3q.)$

Fig. 1. Network $D = (V, E; \omega)$.

The arc set of D is $E = \{a_0, b_0\} \cup A_1 \cup A_2 \cup A_3 \cup B$, $n = |E|$. (See Fig.1) Consider maximum flow game Γ_D on this network D. Obviously, $v(E) = 3|F|$.

Let $x \in R^n$ be a candidate for a core member, which is given by:

$$
\begin{aligned}
x(a_0) &= 2q - \frac{1}{3|F|};\\
x(b_0) &= 3|F| - 3q + (\frac{1}{3|F|} - \frac{1}{3(q+1)});\\
x(e) &= \frac{1}{3|F|(q+1)}, \quad \forall e \in A_1;\\
x(e) &= 0, \qquad \forall e \in A_2;\\
x(e) &= \frac{1}{3}, \qquad \forall e \in A_3;\\
x(e) &= 0, \qquad \forall e \in B.
\end{aligned}
$$

Easily, we have $x(E) = \sum_{e \in E} x(e) = 3|F| = v(E)$. We shall claim that x is not in the core of the maximum flow game Γ_D if and only if F contains an exact cover for X.

Suppose that x is not in the core of Γ_D, and let $S \subset E$ be a coalition such that $x(S) < v(S)$. We would like such an 'unsatisfied' coalition S to exhibit an exact cover.

1). S contains a_0. Otherwise, $x(S) \geq 0 = v(S)$, a contradiction.

2). S does not contain b_0. Otherwise, if $b_0 \in S$, we will discuss in two cases:

Case 1. $|S \cap A_3| = k \leq 3q - 1$.

$$
\begin{aligned}
&v(S) \leq k + (3|F| - 3q),\\
&x(S) \geq (2q - \frac{1}{3|F|}) + [3|F| - 3q + (\frac{1}{3|F|} - \frac{1}{3(q+1)})] + \frac{1}{3}k > v(S).
\end{aligned}
$$

Case 2. $|S \cap A_3| = 3q$.

If $|S \cap A_1| \leq |F| - 1$, then $v(S) \leq 3|F| - 3$,

$$x(S) \geq (2q - \frac{1}{3|F|}) + [3|F| - 3q + (\frac{1}{3|F|} - \frac{1}{3(q+1)})] + \frac{1}{3}3q > v(S).$$

If $|S \cap A_1| = |F|$, then $v(S) \leq 3|F|$,

$$x(S) = (2q - \frac{1}{3|F|}) + |F|\frac{1}{3|F|(q+1)} + [3|F| - 3q + (\frac{1}{3|F|} - \frac{1}{3(q+1)})] + 3q\frac{1}{3} \geq v(S).$$

It is contrary to the fact $x(S) < v(S)$ in both Case 1 and Case 2, so $b_0 \notin S$.

3). S contains all $3q$ arcs in A_3, and $v(S) = 3q$.

If $|S \cap A_3| = k \leq 3q - 1$, then

$$x(S) \geq (2q - \frac{1}{3|F|}) + \frac{1}{3}k \geq k + (\frac{2}{3} - \frac{1}{3|F|}) > k \geq v(S),$$

which is a contradiction. So $|S \cap A_3| = 3q$.

If $v(S) < 3q$, then we have $v(S) \leq 3q - 1$, since $v(S)$ must be integer under the given integral arc capacities. It follows that

$$x(S) \geq (2q - \frac{1}{3|F|}) + 3q\frac{1}{3} > 3q - 1 \geq v(S),$$

also a contradiction. So $v(S) = 3q$.

4). S contains exactly q arcs in A_1. Suppose that $|S \cap A_1| = k$, then

$$x(S) = (2q - \frac{1}{3|F|}) + k\frac{1}{3|F|(q+1)} + 3q\frac{1}{3} = 3q - \frac{1}{3|F|}(1 - \frac{k}{q+1}).$$

Since $x(S) < v(S) = 3q$, we have $k < q + 1$. Again since each arc in A_1 has only 3 arcs in A_2 followed, so we must have $k \geq q$. That is, $k = q$.

Therefore, under our assumption of $x \notin \text{Core}(\Gamma_D)$, the above discussions 1)—4) imply that F contains an exact cover of X.

On the other hand, if F admits an exact 3-cover $F' = \{f_{i_1}, f_{i_2}, \cdots, f_{i_q}\}$, let $S = \{a_0\} \cup \{(v_0, u_{i_k}) : k = 1, 2, \cdots, q\} \cup A_2 \cup A_3$, then

$$v(S) = 3q,$$
$$x(S) = (2q - \frac{1}{3|F|}) + q\frac{1}{3|F|(q+1)} + q < 3q,$$

which implies that x is not in the core of the maximum flow game Γ_D.

Finally, easily to see that the construction of Γ_D and the candidate core member x can obviously be carried out in polynomial time. The proof is completed.

As a direct corollary, we come to the same conclusion for linear production games in Corollary 2.2, that is, the problem of checking membership in the core is also *co-NP*-complete.

Corollary 2.2 It is NP-complete to decide that given a linear production game $\Gamma \equiv (N, v)$ and a vector $x \in R^n$ $(n = |N|)$ with $x(N) = v(N)$, is x not a core member of this game, i.e., does there exist a coalition $S \subset N$ such that $x(S) < v(S)$?

3 Checking Membership of Core for Steiner Tree Game

Theorem 3.1 It is *NP*-hard to decide that given a balanced Steiner tree game $\Gamma(G) = (N, v)$ on weighted graph $G = (V, E; \omega)$ and a vector $x \in R^n$ $(n = |N|)$ with $x(N) = v(N)$, dose there exist a coalition $S \subseteq N$ such that $x(S) > v(S)$?

Proof. Given an arbitrary instance of EXACT COVER BY 3-SETS (X3C): a finite set $X = \{x_1, x_2, \cdots, x_n\}$ and a collection $F = \{f_1, f_2, \cdots, f_{|F|}\}$ of 3-element subsets of X $(q \geq 2, |F| \geq 2q)$. We construct a Steiner tree game $\Gamma(G)$ on weighted graph $G = (V, E; \omega)$ and a candidate vector x for a core member.

The vertex set V of G consists of four parts:

- $N_X = \{v_1, v_2, \cdots, v_{3q}\}$, the vertex $v_i \in N_X$ is corresponding to the element $x_i \in X$, $i = 1, 2, \cdots, 3q$, N_X represents a subset of consumers;
- $N_F = \{u_1, u_2, \cdots, u_{|F|}\}$, the vertex $u_j \in N_F$ is corresponding to the subset $f_j \in F$, $j = 1, 2, \cdots, |F|$, N_F also represents a subset of consumers;
- g_1 and g_2 are two additional vertices, they represent switches;
- v_0 represents central supplier.

Thus, $V = N_X \cup N_F \cup \{g_1, g_2, v_0\}$. Let $N = N_X \cup N_F$, $n = |N|$. (See Fig.2).

The edge set E of G and edge weight function $\omega : E \to R^+$ are defined as follows:

- $a_0 = (v_0, g_1), \quad \omega(a_0) = 2q - 1;$
- $b_0 = (g_1, g_2), \quad \omega(b_0) = q + 1;$
- $E_1 = \{(g_1, u_i) : i = 1, 2, \cdots, |F|\}, \quad \forall e \in E_1 : \omega(e) = q + 1;$
- $E_2 = \{(g_2, u_i) : i = 1, 2, \cdots, |F|\}, \quad \forall e \in E_2 : \omega(e) = q;$
- $E_3 = \{(u_i, v_j) : x_j \in f_i, i = 1, 2, \cdots, |F|, j = 1, 2, \cdots, 3q\},$
  $\forall e \in E_3 : \omega(e) = 2q + 1;$

The edge set $E = \{a_0, b_0\} \cup E_1 \cup E_2 \cup E_3$. (See Fig.2). Denote the Steiner tree game on the weighted graph G by $\Gamma(G) = (N, v)$.

Our construction of graph G is much similar to that in Faigle, et al.[5]. The only difference is the weights of edges in E_3. The weights of edges in E_3 is sufficient large so that each vertex in N_X does not serves as a switch in the minimum Steiner tree of G. For any coalition $S \subseteq N$, denote the minimum Steiner tree w.r.t. $S \cup \{v_0\}$ by $T_S = (V_S, E_S)$. We have

1). If $|V_S \cap N_F| \leq q + 1$, then $E_S \cap E_2 = \emptyset$, and

$$v(S) = \omega(E_S) = (2q - 1) + (q + 1)|V_S \cap N_F| + (2q + 1)|S \cap N_X|. \tag{3.1}$$

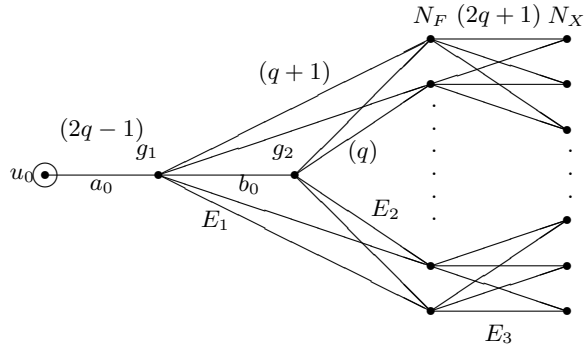
2). If $|V_S \cap N_F| \geq q + 1$, then $E_S \cap E_1 = \emptyset$, and

$$v(S) = \omega(E_S) = (2q - 1) + (q + 1) + q|V_S \cap N_F| + (2q + 1)|S \cap N_X|. \tag{3.2}$$

Therefore, for the grand coalition N, $v(N) = 6q(q + 1) + q|F|$. Let

$$x(u_i) = q + \frac{3q}{3q + |F|}, \quad \forall i = 1, 2, \cdots, |F|;$$

$$x(v_j) = 2q + 1 + \frac{3q}{3q + |F|}, \quad \forall j = 1, 2, \cdots, 3q;$$



$N_F = \{u_i : \ i = 1, 2, \cdots, |F|\}$; $N_X = \{v_j : \ j = 1, 2, \cdots, 3q\}$.
(The characters in brackets are the corresponding edge weights)

Fig. 2. $G = (V, E; \omega)$.

easily to verify that $x \in \text{Core}(\Gamma(G))$, this game is balanced.

Define $x \in R^n$ be a candidate for a core member:

$$x(u_i) = q, \quad \forall i = 1, 2, \cdots, |F|;$$
$$x(v_j) = 2q + 2, \quad \forall j = 1, 2, \cdots, 3q;$$

We shall claim that x is not in $\text{Core}(\Gamma(G))$ if and only if F contains an exact cover for X.

Assume x is not in $\text{Core}(\Gamma(G))$, and let $S \subseteq E$ be a coalition satisfying $x(S) > v(S)$. Then S must possess the following properties: 1). $S \cap N_X \neq \emptyset$; 2). $|S \cap N_F| = q$.; 3). $S \cap N_X = 3q$. Therefore, the assumption of $x \notin \text{Core}(\Gamma(G))$ implies F contains an exact cover of X.

On the other hand, if F admits an exact 3-cover $F' = \{f_{i_1}, f_{i_2}, \cdots, f_{i_q}\}$, let $S = \{u_{i_k} : k = 1, 2, \cdots, q\} \cup N_X$, then $v(S) = 7q^2 + 6q - 1 < 7q^2 + 6q = x(S)$, which implies that x is not in $\text{Core}(\Gamma(G))$. Easily to see that our construction of game $\Gamma(G)$ and the candidate core member x can be carried out in polynomial time, so the proof is finished.

4 A *NP*-Hard Total Balancedness Condition

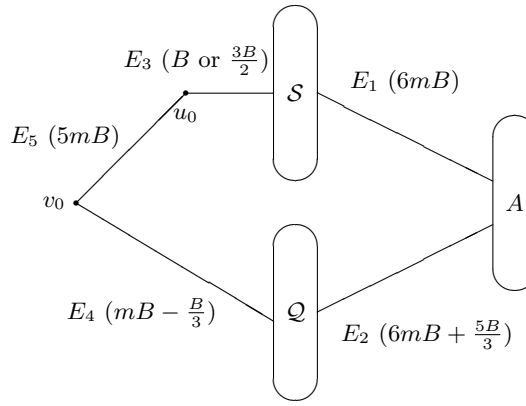
In this section, we prove the problem of testing total balancedness of the Steiner tree game is *NP*-hard. In our proof, we will use 3-PARTITION, which is shown to be *NP*-complete in strong sense in the book of Garey and Johnson [6].

Theorem 4.1 *Testing total balancedness for Steiner tree games is NP-hard.*

Proof. Given any instance of 3-PARTITION: A finite set $= \{a_1, a_2, ..., a_{3m}\}$, a bound $B \in Z^+$, and a *size* : $s(a) \in Z^+$ for each $a \in A$, such that

$$\begin{cases} \frac{B}{4} < s(a) < \frac{B}{2}, \quad \forall a \in A, \\ \sum_{a \in A} s(a) = mB. \end{cases}$$

We let $\mathcal{S} = \{S : S \text{ is a 3-set of } A\}$, $p = |\mathcal{S}| = \mathrm{C}_{3m}^{3}$, and $\mathcal{Q} = \{Q : Q \text{ is a } (3m-1)\text{-set of } A\}$, $q = |\mathcal{Q}| = 3m$.



(The quantities in the brackets are the corresponding edge weights.)

Fig. 3. $G = (V, E; \omega)$

I. First we construct a weighted graph $G = (V, E; \omega)$.

• The node set $V = \{v_0\} \cup N \cup M$ is given as follows:
$\{v_0\}$ is corresponding to the supplier;
$N = A$ is corresponding to the set of consumers;
$M = \mathcal{S} \cup \mathcal{Q} \cup \{u_0\}$ is corresponding to the set of switches.

• The edge set $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$, and the edge weight function $\omega : E \to R^+$ are defined as follows:
$E_1 = \{(a, S) : a \in S, \text{ for all } a \in A, S \in \mathcal{S}\}$, $\forall e \in E_1$, $\omega(e) = 6mB$;
$E_2 = \{(a, Q) : a \in Q, \text{ for all } a \in A, Q \in \mathcal{Q}\}$, $\forall e \in E_2$, $\omega(e) = 6mB + \frac{5B}{3}$;
$E_3 = \{(S, u_0) : S \in \mathcal{S}\}$,

$$\forall e = (S, u_0) \in E_3,\ \omega(e) = \begin{cases} B & \text{if } \sum_{a \in S} s(a) \le B, \\ \frac{3B}{2} & \text{if } \sum_{a \in S} s(a) > B; \end{cases}$$

$E_4 = \{(Q, v_0) : Q \in \mathcal{Q}\}$, $\forall e \in E_4$, $\omega(e) = mB - \frac{B}{3}$;
$E_5 = \{(u_0, v_0)\}$, $\omega(u_0, v_0) = 5mB$.

Denote the corresponding Steiner tree game by $\Gamma(G) \equiv (N, v)$. For convenience of comprehension, we give a sketch of the graph $G = (V, E; \omega)$ in Fig.3. Notice that

1). The weight of each edge $e \in E_1 \cup E_2$ is sufficient large so that in any minimum Steiner tree of G w.r.t. $S \cup \{v_0\}$ ($S \subseteq N$), there are exactly $|S|$ edges in $E_1 \cup E_2$.

2). The construction of edge (u_0, v_0) is to ensure that for any proper subset of A, the vertices in $\mathcal{S}$ are not needed in the corresponding minimum Steiner tree.

II. We show that the cores of all proper subgames of $\Gamma(G)$ are non-empty. Let S be any proper subset of N, $|S| \leq 3m-1$. Assume that $T_S = (V_S, E_S)$ is the minimum Steiner tree of G w.r.t. $S \cup \{v_0\}$, then it must be the case $V_S = \{v_0\} \cup S \cup Q$, where the correponding set $Q \in \mathcal{Q}$ including S. That is, E_S consists of one edge in E_4 and $|S|$ edges in E_2,

$$v(S) = 6mB|S| + \frac{5}{3}B|S| + mB - \frac{1}{3}B := v_S^*. \tag{4.1}$$

Let

$$x(a) = \frac{v_S^*}{|S|} = 6mB + \frac{5B}{3} + \frac{(3m-1)B}{3|S|}, \quad \forall a \in S,$$

easy to verify that it is an element in the core of $\Gamma(G_S)$.

III. We prove that the game $\Gamma(G)$ has non-empty core if and only if A has a 3-partition.

Assume that $T = (V_N, E_N)$ is the minimum Steiner tree of G w.r.t. $N \cup \{v_0\}$, then we have

$$V_N = \{v_0\} \cup N \cup \{S_1, \cdots, S_m\},$$

where $S_1, \cdots, S_m \in \mathcal{S}$ and $\bigcup_{i=1}^m S_i = A$. That is, E_N consists of the edge (u_0, v_0), m edges in E_3 and $3m$ edges in E_1, and

$$v(N) \leq 6mB \cdot 3m + m\frac{3B}{2} + 5mB = 18m^2B + \frac{13}{2}mB := v_N^*. \tag{4.2}$$

In fact,

a). If E_N doesnot contain any edges in $E_1 \cup E_3 \cup E_5$, then it must consist of $3m$ edges in E_2 and two edges in E_4, and $v(N) = 2(mB - \frac{B}{3}) + 3m(6mB + \frac{5B}{3}) > v_N^*$, a contradiction.

b). If E_N contains k $(0 < k < 3m)$ edges in E_2, then it must contain one edge in E_4, and $v(N) \geq (mB - \frac{B}{3}) + k(6mB + \frac{5B}{3}) + 5mB + (3m-k)6mB + \lceil \frac{3m-k}{3} \rfloor B > v_N^*$, also a contradiction.

If A has a 3-partition, then

$$v(N) = 6mB \cdot 3m + mB + 5mB = 18m^2B + 6mB.$$

Let $x(a) = 6mB + 2B$, $\forall a \in N$. By formula (4.1), easy to verify that it is a core member of $\Gamma(G)$.

If A has no 3-partition, suppose the core of $\Gamma(G)$ is not empty and $x \in \text{Core}(\Gamma(G))$. Since $x(N) = v(N) > 18m^2B + 6mB$, there must be a $(3m-1)$-element subset Q^* of N such that

$$x(Q^*) > \frac{3m-1}{3m}(18m^2B + 6mB) = (3m-1)(6mB + 2B).$$

By the formula (4.1), we have

$$v(Q^*) = 6mB(3m-1) + \frac{5}{3}B(3m-1) + mB - \frac{B}{3} = (3m-1)(6mB + 2B) < x(Q^*),$$

which is contrary to our hypothesis that $x \in \text{Core}(\Gamma(G))$.

Therefore, the $\Gamma(G)$ is totally balanced if and only if A has an equivalent 3-partition. Also the construction of $\Gamma(G)$ can be carried out in polynomial time, so testing total balancedness of Steiner tree games is NP-hard.

References

1. I.J. Curiel (1988), Cooperative Game Theory and Applications. Ph.D. dissertation, University of Nijmegen, the Netherlands.
2. X. Deng, T. Ibaraki and H. Nagamochi (1999), Algorithmic aspects of the core of combinatorial optimization games. Mathematics of Operations Research 24, pp. 751–766.
3. X. Deng, T. Ibaraki, H. Nagamochi and W. Zang (2000),Totally balanced combinatorial optimization games, Mathematical Programming 87, pp.441-452.
4. X. Deng and C.H. Papadimitriou (1994), On the complexity of cooperative solution concepts. Mathematics of Operations Research 19, pp. 257–266.
5. U. Faigle, W. Kern, S.P. Fekete and W. Hochstättler (1997), On the complexity of testing membership in the core of min-cost spanning tree games. International Journal of Game Theory 26, pp. 361–366.
6. M.R. Garey and D.S. Johnson (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Francisco.
7. M.X. Goemans and M. Skutella (2000), Cooperative facility location games. SODA.
8. D. Granot and G. Huberman (1981), Minimum cost spanning tree games. Mathematical Programming 21, pp. 1–18.
9. E. Kalai and E. Zemel (1982), Totally balanced games and games of flow. Mathematics of Operations Research 7, pp. 476–478.
10. E. Kalai and E. Zemel (1982), Generalized network problems yielding totally balanced games. Operations Research 30, pp. 498–1008.
11. N. Megiddo (1978), Computational complexity and the game theory approach to cost allocation for a tree. Mathematics of Operations Research 3, pp. 189–196.
12. G. Owen (1975), On the core of linear production games. Mathematical Programming 9, pp. 358–370.
13. L.S. Shapley (1967), On balanced sets and cores. Naval Research Logistics Quarterly 14, pp. 453–460.

Strong Solutions to the Identification Problem

Pino Caballero-Gil and Candelaria Hernández-Goya

Department of Statistics, Operations Research and Computing.
Faculty of Maths, University of La Laguna. Tenerife. Spain
{pcaballe, mchgoya}@ull.es

Abstract. This work proposes two identification algorithms based on some difficult problems. The first scheme is an extremely simple combination of the idea of one-time passwords and the use of public keys. The second proposal allows identification to be implemented without leaking any secret information during the interaction because it combines the two mentioned concepts with a zero-knowledge proof of the possession of a solution to a difficult graph problem through a challenge-response technique.

1 Introduction

Identification schemes are designed to allow one party (the verifier) to gain assurance that the identity of another (the prover) is as declared. The most common technique is the use of secret time-invariant passwords that are shared between both participants. This scheme corresponds to a maximum-disclosure proof and its major security concern is eavesdropping and subsequent replay of the secret information. There are two well-known solutions to this security problem. The weakest of both methods uses one-time shared passwords, whereas the strongest solution consists in applying the idea of challenge-response protocols that allow the prover A (Alice) to convince the verifier B (the system) that she has some information without revealing it. This idea corresponds to minimum-disclosure proofs and has been implemented through several techniques as for example zero-knowledge proofs. This last concept was formalized in 1986 by Goldreich, Micali and Wigderson, [3], who showed that any problem in NP can be given a zero-knowledge proof. This kind of proofs implies zero disclosure, the use of random numbers as challenges and the application of bit commitment schemes. The recent reference [2] provides an extensive review to the area of probabilistic proofs. There we can find assertions such as: "The development of techniques for the construction of efficient zero-knowledge proof systems is still of interest", which refers exactly to the subject of this work.

The work [4] proposes a zero-knowledge proof for the solution to the graph isomorphism problem. Here we use the same problem as basis for the demonstration of the knowledge of a solution to other difficult problems, [1]. Note also that in our proposals there is no need to know how to solve any difficult problem because the user builds an adequate graph for each chosen solution.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 257–261, 2001.
© Springer-Verlag Berlin Heidelberg 2001

This paper is organized as follows: In the following section we comment the main ingredients of our protocols. Sections 3 and 4 present respectively a simple protocol based on one-time passwords and a scheme based on the challenge-response technique. Finally the work ends with some conclusions and open problems.

2 Ingredients

The proposed schemes combine challenge-response techniques with one-time passwords. Now we comment some of the main aspects of these ingredients.

2.1 Challenge-Response

According to the challenge-response technique, the steps followed by the participants A and B can be generally described in the following way:

1. A chooses a random element (witness) of a predefined set, which defines a set of questions (challenges) whose answers (responses) only she knows.
2. B chooses one of the challenges.
3. A provides the corresponding response.
4. B verifies the received response and witness.

If the number of challenges is c, then the probability of A's fraud after m iterations is c^{-m}. Thus, to decrease A's fraud probability, the number m of iterations should be increased. Also, to protect the secret information, A should answer, at the most, to one question for each witness, and never reuse any previous witness. On the other hand, formal zero-knowledge proofs should verify the properties of completeness, soundness and zero-knowledge, [2].

2.2 Graphs

Another important ingredient for these protocols is the difficult graph problem used as basis to guarantee their security. In order to make more efficient the verification phase, this problem should be "one-way", that is to say, it should be such that it is difficult to find a solution but however it is very simple to verify a solution. In these proposals the user has to follow a building process for the construction of an adequate graph from a randomly chosen solution. This process has a different computational complexity depending on the specific graph problem, so this aspect has to be also considered when selecting the graph problem. So, it seems that the most suitable difficult graph problems are those where the solution is a vertex subset, such as the vertex cover, the independent set or the clique problems. Regarding the building process, in both protocols, once chosen a problem P, a number of vertices n and a solution S, the user A has to build a graph G where S is a solution to the problem P. In order to do it, A can apply one of the two following methods:

CM) Constructive method: By adding edges to the null graph N_n

DM) Destructive method: By deleting edges from the complete graph K_n

2.3 Representations

Throughout this work, every secret identification password corresponds to a solution of a difficult graph problem. So, regarding memory requirements, special attention should be paid to the solutions, graphs, functions and problems representations. Every solution is here symbolized by the decimal value of the binary representation where the unitary bits indicate the presence of the corresponding elements of the solution, so in the first protocol, since this is the only object that has to be sent from A to B, the communication complexity is not a great problem because it only implies to send a binary vector whose length depends on the kind of the used problem. On the other hand, the problems have to be published in a certified data base. So, a code dictionary for the list of possible problems can be published in the same directory allowing the use of a short code corresponding to each problem.

3 Non-probabilistic Protocol

Some of the main ideas of the scheme proposed in the following section are introduced now with the definition of the first extremely simple proposal based on one-time passwords. The description of this protocol is as follows:

Precalculation

1. A chooses a graph problem P and an adequate solution, and states the integer S corresponding to this solution
2. A builds a corresponding graph G for P and S
3. A states as her public key the pair (P,G)

Interaction

One-time secret password: A positive integer S

One-time public information: A graph G and a problem P

1. A sends to B the solution S
2. B checks that S is a valid solution according to A's public key (P,G)

This scheme is non probabilistic, so after applying it, there should be no doubt about the identity of the user A thanks to the difficulty of the problem and the well choice of the parameters. Another advantage of this proposal is the no need to share any secret information between both parties, in contrast with other well-known one-time password schemes. Note that with this protocol, once used a solution S, it looses its secret condition, so it is necessary to update it after each identification process. So, during the precalculation phase, A has to carry a building step that is completely independent from previous performances of the identification scheme because the solution changes in every running. In order to make more efficient this building step, in the following section we will propose a probabilistic proof where the user A can take approach of previous performances to build new graphs.

4 Probabilistic Protocol

In this section we present a general interactive challenge-response scheme based on difficult graph problems. Now an informal description is given prior to the formal definition. The protocol can be divided into two parts. The first part is an off-line precalculation of the prover where she chooses her fixed secret identification number S, and her one-time public information: a problem P and a graph G. The second part corresponds to the zero-knowledge procedure composed of m iterations where A commits to certain information, B chooses a challenge, A responses and B verifies the answer and the commitment, so that finally it accepts that A possesses a valid secret identification number. So, the formal description of the algorithm is as follows:

Precalculation

1. A chooses a graph problem P and a suitable solution, and states the integer S corresponding to this solution

2. A builds an adequate graph G where S represents a solution to P

3. A updates the pair (P,G) in the public data base

Interaction

Fixed secret information: A positive integer S

One-time public information: A graph G and a problem P

The following four steps are executed m times:

1. A generates at random an isomorphic copy H of G, and sends it to B jointly with the cipher of both, the isomorphism f and the solution T=f(S) to the problem P in H

2. B obtains from the public data base the information (P,G) associated to the user A and selects randomly a bit b. Depending on its value, it requests her to decipher:

(a)[1] a characteristic question of the solution T, if b=0

(b) the isomorphism f, if b=1

3. A responses to the challenge

4. Depending on the selected challenge, B verifies the corresponding cipher and checks that:

(a) T fulfills the properties of a valid solution to the problem P in the graph H, if b=0

(b) f transforms the graph G in its isomorphic copy H, if b=1

In this scheme there exists certain probability that a fraudulent prover passes the verification. On the other hand, the proof of its completeness, soundness and computational zero-knowledge properties is easily affordable. Note also that with this scheme the solution S does not loose its secret condition, so it is not necessary to change it after each running and there is no need to share any secret information. Furthermore, this probabilistic proof has the advantage of requiring less off-line operations than the previous proposal, provided that the building process can be implemented in a way that for one fixed solution and

[1] The step 2(a) can be replaced by the total disclosure of the solution T when the knowledge of its cardinality does not help to deduce S.

for each different problem, the generation of every graph can take approach of previous generations, or even though it can be easily carried out in parallel. Finally, an important aspect of this protocol is that it requires the assumption of the following hypothesis:

"The knowledge of different pairs of difficult problems and corresponding graphs for which there exists a common solution does not help to solve any of the graph problems".

5 Conclusions and Open Problems

The primary aim of this work is the design of cryptographic protocols in order to solve the identification problem. The schemes here proposed constitute two elegant models that allow the application of many different problems from the Graph Theory. They are the result of the combination of the idea of one-time passwords with public keys in the first case, and commitment schemes with challenge-response in the second. Also the NP-completeness is suggested to be used here with a positive use as a basis for both schemes.

Some questions about the use of the protocols to be recalled are the following:

a) the usefulness of these schemes relies on the great variety of graph problems that can be used as basis for the application of both protocols.

b) the basis problem should be "one-way", that is to say, such that it is difficult to find a solution but however it is extremely simple to verify a solution.

A forthcoming version of this work will include the comparison among the computational efficiency of the algorithms, the search of the most suitable instance graphs and problems, the analysis of minimal assumptions for the implementations and the study of the possible parallelization of the second proposal.

References

1. Garey, M., Johnson, D., Computers and Intractability, Freeman and Company (1979).
2. Goldreich, O., Modern Cryptography, Probabilistic Proofs and Pseudo-randomness. Springer (1999)
3. Goldreich, O., Micali, S., Wigderson, A., How to Prove All NP Statements in Zero-Knowledge and a Methodology of Cryptographic Protocol Design, Advances in Cryptology –CRYPTO '86, 171. Springer-Verlag (1986)
4. Goldwasser, S., Micali, S., Rackoff, C., The knowledge complexity of interactive proof systems, SIAM Journal on Computing, 18, 186-208 (1989)

Area Efficient Exponentiation Using Modular Multiplier/Squarer in GF(2^m)[1]

Hyun-Sung Kim and Kee-Young Yoo

Dept. of Computer Engineering, Kyungpook Nat'l Univ.
Daegu, Korea, 702-701
hskim@ce.knu.ac.kr
yook@knu.ac.kr

Abstract. This paper presents a new exponentiation architecture and multiplier/squarer for GF(2^m), which uses a standard basis representation. The proposed multiplier/squarer is used as kernel architecture of exponentiation. Although the proposed multiplier/squarer computes the multiplication and squaring operations at the same time in GF(2^m), the common parts existing in both operations are only executed once, thereby reducing the required hardware compared to related systolic circuits. The proposed multiplier/squarer can be easily applied to exponentiation architecture. It is also well suited to VLSI implementation because of its regularity, modularity, and unidirectional data flow.

1 Introduction

Finite fields GF(2^m) have found useful in error-correcting codes and cryptography [1][2], however, these applications usually require the computation of multiplications, squares, and exponentiations in GF(2^m). Modular exponentiation is the core computation of numerous public key cryptosystems including RSA. As such, to satisfy real-time requirements, efficient hardware structures for such operations are desirable.

The conventional method for computing squares in GF(2^m) over a standard basis involves a multiplier. Recently, a low latency squarer for GF(2^m) with bi-directional data flow over a standard basis was proposed which involves less chip area and a smaller latency than using a multiplier in GF(2^m). For systolic array implementations of exponentiation architecture for GF(2^m), the architectures in [3] and [8] can be used. The circuit in [3] consists of the 2(m-1) multipliers in [5]; the circuit in [4] requires the m-1 power-sum circuits. The architecture in [3] involves bi-directional data flow, while those in [6] and [7] involve unidirectional data flow.

This paper proposes a new systolic multiplier/squarer and new systolic exponentiation architecture for GF(2^m) with unidirectional data flow over a standard basis. Both circuits involve a smaller latency and less hardware complexity than

[1] This work was supported by grant No. 2000-2-51200-001-2 from the Basic Research Program of the Korea Science & Engineering Foundation.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 262–267, 2001.
© Springer-Verlag Berlin Heidelberg 2001

existing related systems, plus they are well suited to VLSI implementation with fault-tolerant design.

2 Exponentiation Architecture for GF(2^m)

Unlike addition and multiplication, exponentiation in GF(2^m) is very similar to an operation for ordinary integers. Let $A(x)$ and $B(x)$ be elements in GF(2^m), and $P(x)$ be the primitive irreducible polynomial of degree m [5]. The three polynomials $A(x)$, $B(x)$and $P(x)$ are written as

$$A(x)=\sum_{k=1}^{m-1} a_k x^k \ , \ \ B(x)=\sum_{k=1}^{m-1} b_k x^k \ , \ \ P(x)=\sum_{k=1}^{m} p_k x^k , \ \ p_m=1$$

If $e = [e_{m-1}, e_{m-2}, \ldots, e_0]$ is the exponent, a right-to-left algorithm can be performed to compute $A(x)^e$ mod $P(x)$ over GF(2^m). Ordinary right-to-left algorithm in [8] can be slightly changed to Alg. 1, which uses a new multiplication/squaring algorithm, M/S algorithm, as a sub-function for exponentiation algorithm. M/S algorithm will be described in section 3.

```
[Algorithm 1] Exponentiation Algorithm using M/S Alg.
Input  A(x), e, P(x)
Output E(x)=A(x) mod P(x)
Step 1 E(x)=1, T(x)=A(x)
Step 2 for i=0 to m-1 do
Step 3 if e_i==1 (E^(i)(x),T^(i)(x))=M/S(E^(i-1)(x),T^(i-1)(x))
       else(E^(i)(x)=E^(i-1)(x),T^(i)(x))=M/S(E^(i-1)(x),T^(i-1)(x))
```

In step 3, M/S algorithm is called with two parameters $E^{(i-1)}(x)$ and $T^{(i-1)}(x)$, and it returns two computation results of multiplication, $E^{(i-1)}(x)*T^{(i-1)}(x)$, and squaring,

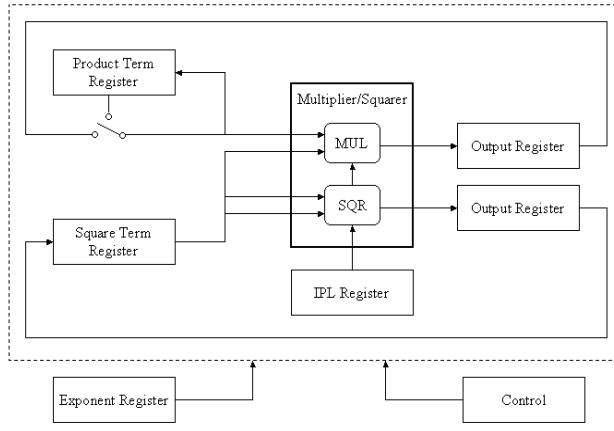


Fig. 1. Proposed exponentiation architecture in GF(2^m)

$T^{(i-1)}(x)*T^{(i-1)}(x)$. Square result is stored to $T^{(i)}(x)$. But, the variable $E^{(i)}(x)$ stores a value depends on the value of e_i. Returned multiplication result is passed to $E^{(i)}(x)$ when e_i has value 1, but previous result, $E^{(i-1)}(x)$, is stored to $E^{(i)}(x)$ in the other case.

Fig. 1 shows a new sequential exponentiation architecture using a new systolic multiplier/squarer, which operates Alg. 1. This architecture includes a multiplier/squarer as its kernel architecture. The multiplier/squarer consists of main two parts, the MUL and the SQR. The MUL operates a multiplication operation whereas the SQR operates a square operation. IPL register keeps an irreducible polynomial. While the multiplier/squarer computes the multiplication operation, it also processes the squaring operation concurrently forming the next square term. The switch depicted in Fig. 1 decides the multiplication result, which chooses the previous result or the current result depends on the exponent.

3 Systolic Multiplier/Squarer for GF(2^m)

The computation steps for the modular multiplication $M(x) = A(x)B(x) \bmod P(x)$ and square $S(x) = A(x)A(x) \bmod P(x)$ are follows as:

$$\begin{aligned} M(x) &= A(x)B(x) \bmod P(x) \\ &= b_0A(x)+b_1[A(x)x \bmod P(x)]+\ldots+b_{m-1}[A(x)x^{m-1} \bmod P(x)] \quad (1) \\ S(x) &= A(x)A(x) \bmod P(x) \\ &= a_0A(x)+a_1[A(x)x \bmod P(x)]+\ldots+a_{m-1}[A(x)x^{m-1} \bmod P(x)] \quad (2) \end{aligned}$$

In the above two equations (1) and (2), [ ] is common to both. Thus, it is much more efficient to compute the common parts only once when multiplication and squaring operations are carried out simultaneously. As such, equation (1) can be reformulated into the recurrence form, called LSB-first form as follows: for $1 \le i \le m$

$$M^{(i)}(x) = M^{(i-1)}(x) + b_{i-1}A^{(i-1)}(x),\ A^{(i)}(x) = A^{(i-1)}(x)\ x \bmod P(x) \quad (3)$$

where $A^{(0)}(x) = A(x)$, $M^{(0)}(x) = 0$, and $M^{(i)}(x) = b_0A(x) + b_1[A(x)\ x \bmod P(x)] + b_2[A(x)\ x^2 \bmod P(x)] + \ldots + b_{i-1}[A(x)\ x^{i-1} \bmod P(x)]$. For $i=m$, $M^{(m)}(x) = M(x) = A(x)B(x) \bmod P(x)$. Note that the two equations in (3) can be performed in parallel.

The above squaring operation (2) can be reformulated to the following LSB-first recurrence form just like equation (3) derived for multiplication: $1 \le i \le m$

$$S^{(i)}(x) = S^{(i-1)}(x) + a_{i-1}A^{(i-1)}(x),\ A^{(i)}(x) = A^{(i-1)}(x)\ x \bmod P(x) \quad (4)$$

where $A^{(0)}(x) = A(x)$, $S^{(0)}(x) = 0$, and $S^{(i)}(x) = a_0A(x) + a_1[A(x)\ x \bmod P(x)] + a_2[A(x)\ x^2 \bmod P(x)] + \ldots + a_{i-1}[A(x)\ x^{i-1} \bmod P(x)]$. For $i=m$, $S^{(m)}(x) = S(x) = A(x)A(x) \bmod P(x)$. Note that the two equations in (4) can also be performed in parallel.

The following bit-wise LSB-first algorithm computing both modular multiplication and squaring simultaneously can be derived from the above equations (3) and (4):

[Algorithm 2] Multiplication and Squaring Algorithm : M/S($E(x)$, $T(x)$)

Input $A(x)=T(x)=(a_{m-1}, a_{m-2}, \ldots, a_0)$
$B(x)=E(x)=(b_{m-1}, b_{m-2}, \ldots, b_0)$

$P(x)=(1, p_{m-1}, p_{m-2}, \ldots, p_0)$
Output $M(x)=A(x)B(x) \bmod P(x), S(x)=A(x)A(x) \bmod P(x)$
Step 1 $M^{(0)}=(M^{(0)}_{m-1}, M^{(0)}_{m-2}, \ldots, M^{(0)}_0)=(0, 0, \ldots, 0)$
$S^{(0)}=(S^{(0)}_{m-1}, S^{(0)}_{m-2}, \ldots, S^{(0)}_0)=(0, 0, \ldots, 0)$
$A^{(0)}=(A^{(0)}_{m-1}, A^{(0)}_{m-2}, \ldots, A^{(0)}_0)=(a_{m-1}, a_{m-2}, \ldots, a_0)$
$A^{(i)}_{-1}=0, 0 \le i \le m$
Step 2 for i=1 to m do
Step 3 for j=1 to m do
Step 4 $M^{(i)}_{m-j}=M^{(i-1)}_{m-j}+b_{i-1}A^{(i-1)}_{m-j}$
Step 5 $S^{(i)}_{m-j}=S^{(i-1)}_{m-j}+a_{i-1}A^{(i-1)}_{m-j}$
Step 6 $A^{(i)}_{m-j}=A^{(i-1)}_{m-j-1}+p_{m-j}A^{(i-1)}_{m-1}$

Fig. 2 shows a DG of Alg. 2 over GF(2^4). The node at position (i, j), ($1 \le i, j \le m$, m = 4) performs three recurrence equations of Alg. 2 in parallel. DG with regular data flows can be easily transformed to a systolic array via a space-time transformation [4].

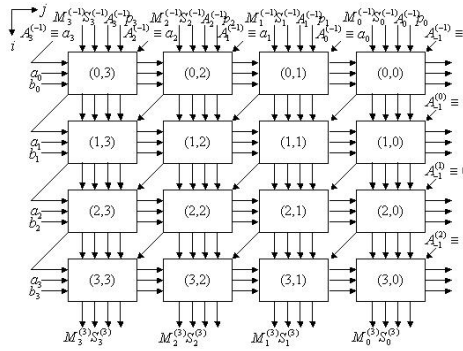
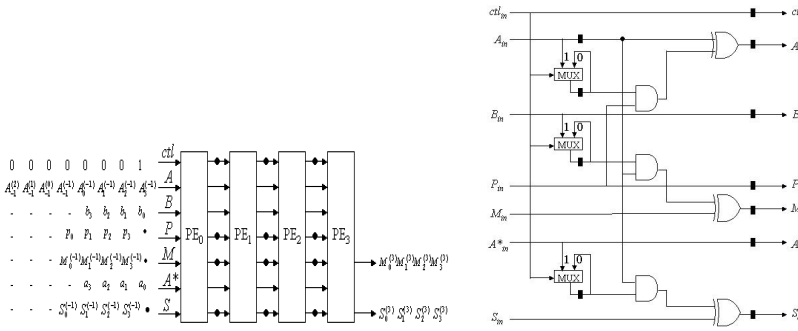


Fig. 2. Dependence Graph for GF(2^4)

The time transformation vector t is equal to [2, 1] because the vector of the equitemporal hyperplane is [1, -2]. The minimum number of PEs to be mapped is when the j column, [0, 1] can be fixed to the projection vector. The space transformation vector s, which is vertical to the projection vector, is [1, 0]. By using the vector s, the nodes on the same row in Fig. 2 can be mapped to the same PE. The proposed systolic multiplier/squarer is shown in Fig. 3.

4 Conclusion

This paper have presented a new systolic multiplier/squarer computing multiplication and squaring operations at the same time, and a new serial-in-serial-out exponentiation architecture for GF(2^m) using a standard basis representation. Although computing multiplication and squaring operations at the same time in GF(2^m), the proposed systolic squarer/multiplier requires less hardware than a dedicated systolic multiplier.



(a) Systolic Multiplier/Squarer over GF(2^4) (b) PE structure

Fig. 3. Proposed Systolic Multiplier/Squarer

Table 1. Comparision of architectures on GF(2^m)

Item	Mul. in Ref. [6]	Mul. in Ref. [7]	Proposed Arch.
No. of PEs	m	m	m
Latency	$3m$	$3m$	$3m$
PE complexity	3 AND gates	2 AND gates	3 AND gates
	2 XOR gates	2 XOR gates	3 XOR gates
	2 MUX(1X2)	2 MUX(1X2)	3 MUX(1X2)
	10 Latches	11 Latches	14 Latches
Critical path	1AND+2XOR+	1AND+1XOR+	1AND+1XOR+
	1MUX+1Latch	1MUX+1Latch	1MUX+1Latch

As shown in Table 1, the PE complexity of the proposed multiplier/squarer was higher than that of existing multipliers. This fact is undeniable because the proposed multiplier/squarer computes modular multiplication and squaring simultaneously, whereas the other multipliers compute only modular multiplication. The proposed hardware is well suited to VLSI implementation because of its regularity, modularity, and unidirectional data flow.

References

1. W. Diffie and M. Hellman, "New Directions in Cryptography", *IEEE Trans. on Info. Theory*, vol. IT- 22(6) pp. 644-654, 1976.
2. I. S. Hsu, I. S. Reed, T. K. Truong, K. Wang, C. S. Yeh, and L. J. Deutsch, "The VLSI implementation of a Reed-Solomon encoder using Berlekamp's bit-serial multiplier algorithm", *IEEE Trans. Computer*, vol. C-33, pp. 906-911. Oct. 1984.
3. S. K. Jain, L. Song, and K. K. Parhi, "Efficient Semisystolic Architectures for Finite-Field Arithmetic," *IEEE Trans. On VLSI Systems*, vol. 6, no. 1, pp. 101-113, 1998.

4. S. Y. Kung, *VLSI Array Processors*, Prentice-Hall, 1987.
5. R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, New York: Kluwer-Academic, 1987.
6. C. L. Wang and J. L. Lin, "Systolic Array Implementation of Multipliers for finite fields GF(2^m)", *IEEE Trans. Circuits Systems*, vol. 38, pp. 796-800, July 1991.
7. C. S. Yeh, I. S. Reed, and T. K. Truong, "Systolic multipliers for finite fields GF(2^m)", *IEEE Trans. Computer*, vol. C-33, pp.357-360, Apr., 1984.
8. P. A. Scott, S. J. Simmons, S. E. Tavares, and L. E. Peppard, "Architectures for Exponentiation in GF(2^m)", *IEEE J. Selected Areas in Comm.*, vol. 6., no. 3, pp.578-586, April., 1988.

A Space Saving Trick for Directed Dynamic Transitive Closure and Shortest Path Algorithms

Valerie King[1] and Mikkel Thorup[1,2]

[1] Department of Computer Science,
University of Victoria, Victoria, BC.
val@csr.uvic.ca, mthorup@research.att.com
[2] AT&T Labs–Research,
Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932.
mthorup@research.att.com

Abstract. We present a simple space saving trick that applies to many previous algorithms for transitive closure and shortest paths in dynamic directed graphs. In these problems, an update can change all edges incident to a node. The basic queries on reachability and distances should be answered in constant time, but also paths should be produced in time proportional to their length. For:

Transitive closure of Demetrescu and Italiano (FOCS 2000)
Space reduction from $O(n^3)$ to $O(n^2)$, preserving an amortized update time of $O(n^2)$.

Exact all-pairs shortest dipaths of King (FOCS 1999)
Space reduction from $\tilde{O}(n^3)$ to $\tilde{O}(n^2\sqrt{nb})$, preserving an amortized update time of $\tilde{O}(n^2\sqrt{nb})$, where b is the maximal edge weight.

Approximate all-pairs shortest dipaths of King (FOCS 1999)
Space reduction from $\tilde{O}(n^3)$ to $\tilde{O}(n^2)$, preserving an amortized update time of $\tilde{O}(n^2)$.

Several authors (Demetrescu and Italiano, FOCS 2000, and Brown and King, Oberwolfach 2000) had discovered techniques to give a corresponding space reduction, but these techniques could be used to show only the existence of a desired dipath, and could not be used to produce the actual path.

1 Introduction

This paper is on saving space for dynamic transitive closure and shortest paths in dynamic directed graphs. For example, the space saving for transitive closure is from $O(n^3)$ to $O(n^2)$. Papers on algorithm are often more concerned with time than space, but a bad space complexity hits like a wall, whereas time only kills slowly. Ignoring for a moment the O-notation, if a computer has gigabyte of memory, with n^3 space, you can only run on graphs with up to $1,000$ nodes, whereas with n^2 space, you can deal with more than $30,000$ nodes. Our space savings techniques are simplifying and tend to decrease rather than increase the constants hidden in the O-notation, both in time and in space.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 268–277, 2001.
© Springer-Verlag Berlin Heidelberg 2001

A fully dynamic graph algorithm is a data structure for a graph which implements an on-line sequence of update operations that insert and delete edges in the graph and answers queries about a given property of the graph. A dynamic graph algorithm should process queries quickly and must perform update operations faster than computing from scratch (as performed by the fastest "static" algorithm).

In this paper, we give dynamic algorithms for weighted directed graphs. The problems considered are as follows, where the queries range over all pairs of vertices u and w:

Transitive closure:
Reachability query: Is there a path from u to w in the current graph?
Path query: Return a path from u to w.

Exact all-pairs shortest paths:
Distance query: What is the distance from u to w?
Path query: Return a path from u to w.

Approximate all-pairs shortest paths:
Distance query: What is an upper bound on the distance from u to w within a factor of $(1+\epsilon)$?
Path query: Return a path from u to w of length within a factor $(1+\epsilon)$ of the distance.

The following update operations are allowed:

- multi-insert(E_v): inserts a set of edges incident to the same vertex v.
- delete(e): deletes any edge currently in the graph.

In this paper, we are only considering amortized time bounds. Deleting an edge is free in that it is paid for by the preceding insertion/initialization of the edge.

We present a simple space saving trick that for all the above problems reduce the space substantially in the current fastest algorithm. More precisely, assuming that the graphs start with an empty edge set, we achieve:

Transitive closure of Demetrescu and Italiano [3,4] Space reduction from $O(n^3)$ to $O(n^2)$, preserving an amortized multi-insert time of $O(n^2)$.

Exact all-pairs shortest dipath of King [9] Space reduction from $\tilde{O}(n^3)$ to $\tilde{O}(n^2\sqrt{nb})$, preserving an amortized multi-insert time of $\tilde{O}(n^2\sqrt{nb})$, where b the maximal edge weight.

Approximate all-pairs shortest dipaths of King [9] Space reduction from $\tilde{O}(n^3)$ to $\tilde{O}(n^2)$, preserving an amortized multi-insert time of $\tilde{O}(n^2)$.

The above update times should be compared with the best static algorithms. Transitive closure and approximate shortest paths can be computed essentially as fast matrix multiplication [8,10,13], which takes $\tilde{O}(n^{2.376})$ time, and for exact all pairs shortest paths, can be computed in $\tilde{O}(n^{2.575})$ time, again using fast

matrix multiplication. Thus, the amortized update times may not seem that impressive. However, fast matrix multiplication is considered rather slow, and it is an open problem to construct a combinatorial algorithm just for transitive closure with sub-cubic running time. Our algorithms are purely combinatorial, and in this view, the dynamic speed-up is optimal in that we can construct a graph from scratch with a multi-insert from each vertex, paying a total cost of $n \times O(n^2) = O(n^3)$ to construct the transitive closure, thus matching the best static combinatorial time bound. Our reduction of the space from $O(n^3)$ to $O(n^2)$ adds optimality with respect to space.

Our ideas also apply to the older single source algorithms of Even and Shiloach [5] and of Ramalingam and Reps [11], improving their internal space from $O(m)$ to $O(n)$. Hence, if used for all pairs shortest paths, the improvement is from $O(mn)$ to $O(n^2)$. We are here particularly interested in Ramalingam and Reps' algorithms because it has proved efficient in practice [7,6]. Our idea also simplifies the implementation which typically reduces the worked performed.

A common theme of all the above algorithms is that they keep tables of distance information along with witnesses. A table entry can only increase if all its witnesses are lost. This theme goes back to Even and Shiloach's decremental single shortest path algorithm [5], and is also used in the fully-dynamic single source shortest path algorithm of Ramalingam and Reps [11]. Unfortunately, there may be many witnesses for each entry, and it is the storage of these witnesses that cause a prohibitive space-overhead.

Recently space improvement similar to ours were reported by Brown and King [2] and of Demetrescu and Italiano [4]. Their idea was to maintain a counter of the number of witnesses for each entry. Again, the entry can only increase if its counter goes to 0. The problem in their solution is that they loose track of witnesses, and then they cannot answer path queries.

Our challenge of identifying witnesses rather than just knowing their existence is similar in spirit and motivation to that of finding witnesses for Boolean matrix multiplication as done by Alon et al. [1], though the techniques we apply here are combinatorial and completely different.

Our contribution is to observe that if we are careful about the order in which we scan for witnesses, then we only need to store the first witness that we meet. When a witness is lost, we just continue the scan for the next witness. Our problem is then to identify scanning orders and properly define witnesses so that the same potential witnesses are not scanned repeatedly.

Contents First, in §2, we present the simplest version of the trick, applying it to Even and Shiloach's classic algorithm [5]. In §2, we also point out the modification needed for Ramalingam and Reps' algorithm [11]. Next, in §3, we show how to tailor the trick to King's exact shortest paths and transitive closure algorithms [9]. For space reasons we defer the treatment of King's exact and approximate shortest paths and Ranmalingam and Reps shortest paths algorithm to the journal version of this paper.

Notation We working on a dynamic directed weighted graph $G = (E, V)$ with edge weight function $\ell : E \to \mathbb{N}$. For any subgraph H of G, $V(H)$ is its vertex set and $E(H)$ its edge set. For any pair of vertices (v, w), $\mathsf{dist}(v, w)$ denotes their distance in G. For each vertex v, $in(v)$ and $out(v)$ denotes its incoming and outgoing edges in G.

2 Even and Shiloach's Algorithm

Our basic idea is most easily presented in terms of Even and Shiloach's decremental single shortest path algorithm [5]. Their result is that we can maintain distances up to some threshold Δ in $O(m\Delta)$ total time. In [5] they just consider unweighted graphs and threshold distance $\Delta = n$, giving them an $O(mn)$ bound for maintaining all distances from the source. We generalize this to weighted graphs.

For a given source node s, the Even-Shiloach algorithm maintains a *shortest path graph* H_s which is the union of all shortest paths from s of length at most Δ. For each vertex v, we maintain its distance $d_s(v)$ from s, and, for each distance $i = 0, ..., \Delta$, we maintain the set of vertices v with $d_s(v) = i$. During our response to an edge deletion, $d_s(v)$ will be incremented by one at a time until it reaches the new distance from s to v.

In the shortest path graph H_s, each edge (u, v) is a *witness* of the current distance to v, in the sense that $\mathsf{dist}(s, v) = d_s(v) = d_s(u) + \ell(u, v) = \mathsf{dist}(s, u) + \ell(u, v)$. Hence, when an edge (u, v) is deleted, it affects distances only if (u, v) is in H_s, and there is no other edge (x, v) to v in H_s.

Suppose the deleted edge (u, v) is a last witness to v in H_s. This implies that $d_s(v)$ has increased, and the outgoing edges (v, w) are no longer valid witnesses of $d_s(w)$. A cascading effect is started: when a node w loses all its incoming edges in H_s, we must delete all its outgoing edges in H_s.

To update H_s efficiently, the cascade is performed one distance i at the time. Let (u, v) be the original edge deleted. Starting with $i \leftarrow d_s(v)$, we do as follows. For each vertex w with $d_s(w) = i$ and no incoming edges in H_s, we remove all its outgoing edges in H_s, and increment $d_s(w)$ by one. This preserves $d_s(w) \leq \mathsf{dist}(s, w)$, and now, for all u with $d_s(u) \leq i$, we know that $d_s(u) = \mathsf{dist}(s, u)$. Next, for each w with $d_s(w)$ increased, we consider all its incoming edges (x, w) in G. The edge (x, w) witnesses $d_s(w) = \mathsf{dist}(s, w)$ if $d_s(x) \leq i$ and $d_s(x) + \ell(x, v) = d_s(w)$, and if so, (x, w) is added to H_s. If no such witness is found, we still have $d_s(w) < \mathsf{dist}(s, v)$. We now set $i \leftarrow i + 1$ and repeat. We terminate when $i = \Delta + 1$ or when we get to a distance i where all nodes have incoming edges in H_s.

For a detailed description and analysis of the above algorithm, the reader is referred to [5]. The running time is dominated by the scanning for witnesses; each time we increment $d_s(w)$, we scan all its incoming edge to see if they witness the new distance. Since this can happen at most Δ times for each w, the total cost is $O(\Delta \cdot \sum_{w \in V(G)} |\{(x, w) \in E(G)\}|) = O(\Delta m)$.

2.1 Our Space Saving Trick

The internal space of the Even-Shiloach algorithm, i.e. the space beyond the input graph G, is $O(m)$ for storing the shortest path graph H_s. We will reduce this to $O(n)$. If we want a decremental data structure for each node, then the space is reduced from $O(mn)$ to $O(n^2)$.

For each vertex v, its list $in(v)$ of incoming edges in G stores the incoming edges in some order. Instead of storing all of H_s, for each v, we store only the edge (x, v) from H_s which is first in $in(v)$. We then have the unique shortest path tree T_s from s, spanning all nodes within distance Δ and such that the parent pointer of each node v is the first edge in $in(v)$ which is on a shortest path from the source to v.

The deletion of an edge (u, v) only has a consequence if (u, v) is in T_s. In that case, we search for a replacement witness. Starting from the successor of (u, v), we scan the edges in $in(v)$, stopping if we reach a new witness (x, v) with $d_s(x) + \ell(x, v) = d_s(v)$. If that happens , we replace (u, v) with (x, v) in T_s, and stop. Else we conclude that v has also lost its last incoming edge in H_s. As in the Even-Shiloach algorithm, each incoming edge to each vertex v is considered at most once for each value of $d_s(v)$. Hence, we preserve the running time of $O(\Delta m)$. However, since T_s is a tree, we have reduced the internal space from $O(m)$ to $O(n)$. Also we save work when edges from $H_s \setminus T_s$ are deleted.

3 King's Algorithms

As the basis for all the fully-dynamic algorithms of King in [9], we want to maintain all distances up to some threshold Δ. Besides edge deletions, we have multi-inserts that for each vertex v can add any set of edges incident to v. The amortized cost per multi-insert is $O(\Delta n^2)$. This also pays for all deletions. The space is $O(n^3)$, but we reduce it to $O(\Delta n^2)$.

The algorithm uses the Even-Shiloach algorithm to maintains $2n$ deletions-only data structures for distances up to Δ. For each vertex v, we have one $Out(v)$ for distances from v, i.e. with v as source, and one $In(v)$ with distances to v, i.e. with v as a sink. For each pair (u, w) of vertices, $\mathsf{d}ist_v(u, w)$ is the distance from u to v in $In(v)$ plus the distance from v to w in $Out(v)$.

We start with a graph with no edges. Whenever we make a multi-insert of edges incident to v, we reinitialize the pair $(In(v), Out(v))$ at amortized cost $O(\Delta m)$. When an edge is deleted, it is deleted from all pairs $(In(v), Out(v))$ created after it was inserted.

Lemma 1. *If the distance from u to w is at most Δ, $\mathsf{d}ist(u,w) = \min_v \mathsf{d}ist_v(u,w)$.*

Proof. Obviously, $\mathsf{d}ist(u, w) \leq \mathsf{d}ist_v(u, w)$ for all v. Consider an arbitrary shortest path from u to w and let v be the last vertex updated on this path. Then all the edges of this path are contained in $(In(v), Out(v))$ so $\mathsf{d}ist_v(u, w) = \mathsf{d}ist(u, w)$.

Thus, our problem is to keep track of the vertex v minimizing $\mathsf{d}ist_v(u,w)$ for each pair (u,w) of vertices. King's idea is, for each possible distance $d \in \{1,...,\Delta\}$, to maintain the list $L(u,w,d)$ of vertices v with $\mathsf{d}ist_v(u,w) = d$.
The King algorithm: Whenever a vertex u increases its distance to v in $In(v)$, this increases all distances $\mathsf{d}ist_v(u,w) \leq \Delta$. If d and d' are the old and the new distance of v, v has to be moved from $L(u,w,d)$ to $L(u,w,d')$. This move takes constant time. Each vertex u increases its distance at most Δ times in $In(v)$, and each time, we have to perform a move between lists for each vertex w with $\mathsf{d}ist_v(u,w) \leq \Delta$. Thus, u pays a total cost of $O(\Delta n)$ for increases to its distance in $In(v)$. The cost for increases to its distance in $Out(v)$ is the same. Thus, for each pair $(In(v), Out(v))$, the total cost of distance increases is $O(\Delta n^2)$, which is attributed to the multi-insert creating $(In(v), Out(v))$.

To get the distance matrix, for each (u,w), we maintain the smallest d for $L(u,w,d) \neq \emptyset$. Then $d = \mathsf{d}ist(u,w)$. For any vertex v in $L(u,w,d)$, the pair $(In(v), Out(v))$ provides a shortest path from u to w. When doing a multi-insert, the above is trivially done by scanning all lists in $O(n^2\Delta)$ time. When doing deletions, our amortization argument allows the multi-inserts to pay for all distance increases, even if these are considered to be done one at the time. If $L(u,w,d)$ becomes empty because some vertex v is moved to $L(u,w,d')$, we can afford to scan all the lists $L(u,w,e)$, $e = d+1,...,d'$ by charging the cost to the multi-inserts.

Each pair $(In(v), Out(v))$ takes $O(m)$ space. For each (u,v,w), v is in $L(u,w,\mathsf{d}ist_v(u,w))$, so the total number of elements in the lists is $O(n^3)$. Moreover, there are $O(\Delta n^2) = O(n^3)$ lists. Hence, the total size of the data structure is $O(n^3)$.

3.1 Improving the Space

To reduce the space of the above data structure to $O(\Delta n^2)$, we need to address both the pairs $(In(v), Out(v))$ and the witness lists $L(u,w,d)$.

$(In(v), Out(v))$ For each pair $(In(v), Out(v))$, we wish to reduce the space to $O(n)$ using the trick from §2.1. However, that trick assumed for each vertex w, that we could traverse $in(w)$ in some fixed order. That is, if (x,w) was the witness of w in T_v, and if (x,w) was deleted, we could continue the traversal for witnesses from the successor of (x,w) in $in(w)$ knowing that the same edge would never be considered twice for the same distance $d_v(w)$.

To deal with the differing incidence lists, we spend $O(\Delta n^2)$, rather than $O(\Delta m)$, to maintain each pair $(In(v), Out(v))$. This does not affect our asymptotic cost, since we have already incurrred this cost for each multi-insert. Hence we can spend $O(n)$ time to traverse $in(w)$ in $Out(v)$.

We fix one arbitrary global ordering $\prec$ of all the vertices. For each vertex w its list $in(w)$ in $Out(v)$ will be ordered as $\prec$ orders its predecessors. However, $in(w)$ will be stored only implicitly. We keep a global $n \times n$ incidence matrix which for each $(x,y) \in V^2$ which stores its length $\ell(x,y)$ or ∞ if (x,y) is not an edge.

We traverse $in(w)$ in $Out(v)$ by going through all the vertices $x \in V$ in the order $\prec$ and keeping a pointer to our current location in the order. For each x, we check if (x, w) is an appropriate witness, i.e., if $\mathsf{d}ist_v(v, x)+\ell(x, w) = dist_v(v, w)$. For each edge (x, w) which serves as a witness, we keep a pointer back to v, so that when (x, w) or $\mathsf{d}ist_v(v, x)$ is changed, we know to look for another witness.

We only increment $\mathsf{d}ist_v(v, w)$ after the order is traversed. We only decrement $\mathsf{d}ist_v(v, w)$ when $(In(v), Out(v))$ is reinitialized. At botht times, we reset the pointer to the beginning of the order. Note that in this space-saving version, we may inadvertently add a witness which was created after the last time $Out(v)$ was initialized.

We maintain a shortest path tree of edge witnesses as described in Section 2.1, so that nodes affected by a distance increase can be easily determined.

As claimed, the total time to traverse the order is $O(n)$, giving the amortized cost of $O(\Delta n^2)$ for each pair $(In(v), Out(v))$. Our total space for all the pairs $(In(v), Out(v))$, including the global incidence matrix, is now $O(n^2)$.

***L*(*u*, *w*, *d*)** Instead of the lists $L(u, w, d)$ of vertices v with $\mathsf{d}ist_v(u, w) = d$, we maintain only the oldest witness v with $\mathsf{d}ist_v(u, w) \leq d$. This reduces the space to a constant per list, hence to to $O(\Delta n^2)$, as desired.

The switch from '=' to '$\leq$' is crucial to our amortization. The point is that if v is not a witness for $\mathsf{d}ist_v(u, w) \leq d$, it will not become so before $(In(v), Out(v))$ is replaced by a multi-insert.

We maintain a historical list of the vertices ordered according to dates. Note that this order has nothing to do with the order $\prec$ used previously. When we do a multi-insert of edges around v, we move v to the end of the historical list.

Suppose $d = \mathsf{d}ist_v(u, w)$ is increased. If v was the oldest witness of (u, w, d), we scan the historical list starting from the successor of v, stopping as soon as we find a younger vertex v' with $\mathsf{d}ist_{v'}(u, w) = d$. Then v' is the new oldest witness of (u, w, d), and otherwise, there is no witness. If $\mathsf{d}ist_v(u, w)$ was increased by more than one, we repeat the above scanning if v was also the oldest witness for $(u, w, d + 1)$. As soon as we reach some value e so that v was not the oldest witness for (u, w, e), we stop; if $v' \neq v$ is the oldest witness for (u, w, e), it is also an older witness for all (u, w, f) with $f > e$.

Since we always scan younger and younger witnesses, we see that a vertex v, between multi-inserts around v, can only be scanned once for each triple (u, w, d). So, the amortized cost of each multi-insert is $O(\Delta n^2)$. By getting rid of most pointers, we also reduce the constants hidden in the O-notation.

Below, we briefly describe how the above techniques are applied by King [9] to transitive closure, and leave the discussion of approximate and exact shortest paths to the full paper. We show that our space improvements remain valid for these applications.

3.2 Transitive Closure

King maintains the transitive closure of a digraph, in a manner similar to the classical method of repeated squaring of matrices. For $i = 0, 1, ..., h = \lceil \log n \rceil$,

we maintain a digraph G^i that is a subgraph of the transitive closure, and which is guaranteed to have an edge (v, w) if there is a dipath from v to w in G of length at most 2^i, i.e., $G^0 = G$, and G^h is the transitive closure.

For $i > 0$, and for each vertex $v \in V$, we maintain the pair $(In^{i+1}(v'), Out^{i+1}(v'))$, with $\Delta = 2$. For $i > 0$, the edges of G^{i+1} are defined to be all pairs (u, w) such that there is a path from u to w in $(In^i(v), Out^i(v))$ of length 3 or less.

Lemma 2 (9). *If a dipath P from u to w has length at most 2^{i+1} and the youngest vertex in the dipath is v, there is a path from u to w in $(In^{i+1}(v), Out^{i+1}(v))$ of length 3 or less.*

Proof. Let v' be a middle vertex in P. Assume v' is in $Out^{i+1}(v)$. Then $P[u, v]$, $P[v, v']$, $P[v', w]$ are all of length at most 2^i. Hence (u, v), (v, v') and (v', w) were all in G^i when $(In^{i+1}(v), Out^{i+1}(v))$ was last reinitialized.

To save space, we implement the $(In^{i+1}(v'), Out^{i+1}(v'))$ pairs as in the previous section, with incidence matrices for $i = 1, ..., h$. When a multi-insert of edges around v occurs, and G^0 has been updated, then for $i = 1, ..., h$, we reinitialize only $(In^{i+1}(v), Out^{i+1}(v))$ using all edges currently in G^i. A the multi-insert may create edges in G^i not incident to v which do not affect older pairs $(In^{i+1}(v'), Out^{i+1}(v'))$.

If an edge is deleted from G, it is deleted from the incidence matrix for G^0. For $i = 1, ..., h$, as edges in G^{i-1} are deleted, this may result in the destruction of paths in $(In^{i+1}(v'), Out^{i+1}(v'))$ which in turn may result in the deletion of edges in G^{i+1}.

Given any edge $(u, w) \in G^{i+1}$, to expand it to a path, we consider the witness v for $\mathsf{dist}_v^{i+1}(u, w) \leq 3$. Then $(In^{i+1}(v), In^{i+1}(v))$ provide us a path of length ≤ 3 from (u, w) using edges from G^i, and each of these edges can be expanded recursively.

To get simple paths, we combine with a linear time computation of strongly connected components [12]. For each strongly connected component, using an in-tree and an out-tree from an arbitrary vertex, we provide a simple path between any pair of vertices in the same strongly connected component. In linear time, we also label each vertex with the strongly connected component it is in. Now we only need to reconstruct segments between strongly connected components. So, if G^{i+1} provides us with a path of edges from G^i, we only expand edge (x, y) if x and y are in different connected components.

Since there are $\log n$ levels, we maintain the transitive closure in $O(n^2 \log n)$ amortized time per multi-insert and $O(n^2 \log n)$ space.

4 Demetrescu-Italiano Transitive Closure

The Demetrescu-Italiano transitive closure algorithm represents the graph as an adjacency matrix and achieves a $\log n$ speed-up by basing its algorithm on the static recursive scheme [8,10] for computing transitive closure, rather than

repeated squaring. The adjacency matrix of the graph is subdivided into four sub-matrices. The transitive closure can be shown to be a polynomial over these four sub-matrices. [4,3] show that the problem of updating a polynomial over Boolean matrices can be reduced to updating the product of degree 2 polynomials, with some errors allowed, as described below. Solving this problem in $O(n^2)$ space gives an $O(n^2)$ space bound for the whole algorithm.

Let M_1 and M_2 be two $n \times n$ Boolean matrices. In computing their product, there are two kinds of updates allowed to either matrix in which 0's are changed to 1's: one where any entry can be set to 1 and one where just entries in a particular row i and column i can be set to 1. The first is done by LazySet, the second by SetRow(i, M) and SetCol(i, M), respectively. In addition, 1's can be changed to 0's by ReSet. The "error" allowed is that in certain circumstances, the bits set by LazySet can be ignored.

When entries (x, y) in M_1 and (y, z) in M_2 are both 1, we call these entries a witness pair for (x, z), and y a witness for (x, z). A witness pair (x, z) cannot be ignored if its two entries (x, y) and (y, z) exist currently and they existed since a time when one of them was contained in a row or column updated by a SetRow or SetCol. That is, the problem is to maintain for each entry in the matrix product a witness pair (x, y, z) if one exists and was created at or before the most recently executed $SetRow(x, M_1)$, $SetCol(y, M_1)$, $SetRow(y, M_2)$, or $SetCol(z, M_2)$.

The goal is to spend no more than $O(n^2)$ time per SetCol and SetRow. Essentially, LazySet does nothing except change bits in M_1 and M_2; SetCol and SetRow do the work. The cost of ReSet is amortized against the operations which change the 0's to 1's.

The implementation of this in $O(n^3)$ space is straightforward, and involves the keeping of lists of relevant witnesses.

Our result: We can accomplish the multiplication of two matrices (with the allowable errors)more easily while maintaining witnesses. We do this with no change to the asymptotic time, maintaining a cost of $O(n^2)$ time per SetRow and SetCol, and space $O(n^2)$. We use the technique of the previous sections, of keeping a pointer to the next available witness. We keep:

– For each x, z, a pointer $P1(x, z)$ which is initially set to the first column of y in M_1 such that $M_1(x, y)$ AND $M_2(y, z)$; these two entries are a witness pair for (x, z)

– For each x, z, a pointer $P2(x, z)$ which is initially set to the first row y in M_2 such that $M_1(x, y)$ AND $M_2(y, z)$; these are a witness pair for (x, z).

– A queue of y 's ordered by order of last execution of either SetCol(y, M_1) or SetRow(y, M_2).

– For each (x, z), a pointer $P3(x, z)$ which is initially set to the first y in the queue such that $M_1(x, y)$ AND $M2(y, z)$

– Back pointers from the entries which are witnesses back to the (x, z) for which they are a witness.

To do updates:

When SetRow(x) in M_1 is executed, we reset $P1(x, z)$ to the beginning for all z.

When SetCol(z) in M_2 is executed, we reset $P2(x, z)$ to the beginning for all x. When $M_1(x, y)$ or $M_2(y, z)$is set to 0 during a ReSet:
– If $P1(x, z)$ points to y, then we move $P1(x, z)$ forward until another witness is found or the end of the row.
– If $P2(x, z)$ points to y then we move $P2(x, z)$ forward until another witness is found or the end of the column.
– If $P3(x, z)$ points to y, then we advance $P3(x, z)$ until another witness is found.
Running time analysis: Each time a pointer is set back to the beginning of a row or column, there may be another n checks at a cost of $O(1)$ each. $P3(x, z)$ is never set back but there may be an added cost of O(1) for every pair (x, z) for each queue change. We charge the SetCol or SetRow to cover this.

References

1. N. Alon, Z. Galil, O. Margalit, and M. Naor. Witnesses for boolean matrix multiplication and for shortest paths. In *Proceedings of the 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 417–426, 1992.
2. G. Brown and V. King. Space-efficent methods for maintaining shortest paths and transitive closure: Theory and practice, August 2000. Presentation at Workshop on Efficient Algorithms at Oberwolfach.
3. C. Demetrescu. *Fully Dynamic Algorithms for Path Problems on Directed Graphs.* PhD thesis, Dip. Informatica e Sistemistica Universita' di Roma "La Sapienza", 2000.
4. C. Demetrescu and G. Italiano. Fully dynamic transitive closure: Breaking through the $o(n^2)$ barrier. In *Proc. 41st IEEE Symp. on Foundations of Computer Science*, pages 381–389, 2000.
5. S. Even and Y. Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981.
6. Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proc. 19th IEEE INFOCOM - The Conference on Computer Communications*, pages 519–528, 2000.
7. D. Frigioni, M. Ioffreda, U. Nanni, and G. Pasqualone. Experimental analysis of dynamic algorithms for the single-source shortest path problem. *ACM Jounal of Experimental Algorithmics*, 3, article 5, 1998.
8. M. E. Furman. Application of a method of rapid multiplication of matrices to the problem of finding the transitive closure of a graph. *Soviet Math. Dokl.*, 11(5):1250, 1970.
9. V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. 40th IEEE Symp. on Foundations of Computer Science*, pages 81–89, 1999.
10. I. Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971.
11. G. Ramalingam and T. W. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Jounal of Algorithms*, 21(2):267–305, 1996.
12. R. E. Tarjan. Deep-first search and linear graph algorithms. *SIAM J. Computing*, 1:146–160, 1972.
13. U. Zwick. All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms. In *Proceedings of the 39rd IEEE Annual Symposium on Foundations of Computer Science*, pages 310–319, 1998.

Finding the Most Vital Node of a Shortest Path*

Enrico Nardelli[1,2], Guido Proietti[1,2], and Peter Widmayer[3]

[1] Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila,
Via Vetoio, 67010 L'Aquila, Italy.
{nardelli,proietti}@univaq.it.
[2] Istituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche,
Viale Manzoni 30, 00185 Roma, Italy.
[3] Institut für Theoretische Informatik, ETH Zentrum,
CLW C 2, Clausiusstrasse 49, 8092 Zürich, Switzerland.
widmayer@inf.ethz.ch.

Abstract. In an undirected, 2-node connected graph $G = (V, E)$ with positive real edge lengths, the distance between any two nodes r and s is the length of a shortest path between r and s in G. The removal of a node and its incident edges from G may increase the distance from r to s. A *most vital node* of a given shortest path from r to s is a node (other than r and s) whose removal from G results in the largest increase of the distance from r to s. In the past, the problem of finding a most vital node of a given shortest path has been studied because of its implications in network management, where it is important to know in advance which component failure will affect network efficiency the most. In this paper, we show that this problem can be solved in $O(m + n \log n)$ time and $O(m)$ space, where m and n denote the number of edges and the number of nodes in G.

1 Introduction

The computational infrastructure throughout society is becoming increasingly large and complex. Networks of workstations are vulnerable to attack and failure, and it is generally recognized that the survivability of our computing systems is a critical issue (IEEE Computer dedicated its August 2000 issue to this topic). We are interested in a particular type of survivability: How is a communication network affected by the failure of a component? In this paper, we consider the effect that a node failure will have on a shortest path between two nodes. Our scenario assumes that each message is routed along a shortest path in a communication graph from its source to its destination. When a node on that path fails, we need to replace the old route by a new one, preferably by a shortest path in the graph that does not contain the failed node. Let us call this new route a *replacement (shortest) path*; it will in general be longer than the path

* This work has been partially supported by the Research Training Network contract no. HPRN–CT–1999–00104 funded by the European Union, and by the Research Project REACTION, partially funded by the Italian Ministry of University.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 278–287, 2001.
© Springer-Verlag Berlin Heidelberg 2001

it replaces, but it certainly will never be shorter. From a network management point of view, it is desirable to know for a shortest path ahead of time which node failure will result in the longest replacement path. Such a node is called a *most vital node*, because its failure degrades the transmission from the source to the destination most strongly.

The problem of finding a most vital node of a shortest path has been defined and motivated by Corley and Sha [3]. More precisely, they considered the more general problem of finding the k most vital nodes of a shortest path, that is the k nodes whose removal will increase the distance between the source and the destination node the most, and they gave an exponential algorithm for solving the problem. For the case $k = 1$, an efficient implementation of their algorithm requires $O(mn + n^2 \log n)$ time, where m and n denote the number of edges and the number of nodes in the underlying graph. Note that this is not better than the trivial bound that we get by recomputing from scratch the replacement shortest path for every node along the given shortest path. Later on, Bar-Noy et al. [2] proved that, for arbitrary k, the problem is strongly NP-hard. Finally, Venema et al. [13] studied the problem for $k = 1$ in a parallel computing environment, providing a polynomial algorithm.

In a related scenario, nodes are reliable, but edges can fail. The problem of finding a *most vital edge* on a shortest path has been studied extensively in the past: We look for an edge whose failure leads to the longest replacement path [1,2,3]. Now, naturally a replacement path is just a path avoiding the failed edge. Let us assume that the source and destination nodes lie in a 2-edge connected component of the given graph; otherwise, a most vital edge is just a bridge between these nodes, and that is easy to find. The fastest algorithm to compute most vital edges on a pointer machine runs in $O(m+n \log n)$ time and $O(m)$ space [6]. On a RAM, there is an algorithm to solve the problem (and an interesting variant of it defined in [7]) in $O(m \cdot \alpha(m, n))$ time and $O(m)$ space [8], where $\alpha(m, n)$ denotes the functional inverse of the Ackermann function defined in [12].

In this paper, we show that the problem of finding a most vital node for a shortest path in a 2-node connected, undirected and positively weighted graph can be solved on a pointer machine in $O(m + n \log n)$ time and $O(m)$ space. The efficiency of our algorithm is based on two considerations. First, we make use of specific structural properties of replacement paths in the computation. This is realized by means of a priority queue that stores certain distance values for certain nodes. Unfortunately, the priority queue contains distance values that would lead to an incorrect result if they ever would be used, and it also contains the desired distance values leading to the correct result. The reason for this mix is algorithmic performance: We have no way of efficiently distinguishing between both, but we make sure that the algorithm never uses the undesired values. Second, we perform this computation incrementally as we visit the nodes along the shortest path.

The paper is organized as follows: In Section 2 we define the problem formally and give the required basic definitions. In Section 3, we present the structural

properties of replacement paths, along with our algorithm. Finally, Section 4 discusses modifications and further applications, and lists some open problems.

2 Basic Definitions

Let $G = (V, E)$ be an undirected graph, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. Let n and m denote the number of nodes and the number of edges, respectively, and, for each $e \in E$, let $w(e)$ be a positive real length. A graph $H = (V(H), E(H))$ is called a *subgraph* of G if $V(H) \subseteq V$ and $E(H) \subseteq E$. If $V(H) = V$ then H is called a *spanning subgraph* of G.

A *simple path* (or a *path* for short) in G is a subgraph P with $V(P) = \{v_0, v_1, \ldots, v_k | v_i \neq v_j \text{ for } i \neq j\}$ and $E(P) = \{(v_i, v_{i+1}) | 0 \leq i < k\}$, also denoted as $P(v_0, v_k) = \langle v_0, \ldots, v_k \rangle$. Path $P(v_0, v_k)$ is said to go from v_0 to v_k or, alternatively, to be between v_0 and v_k. Its *length* is the sum of the lengths of the path edges, and will be denoted as $|P(v_0, v_k)|$. A graph G is *connected* if, for any two nodes $u, v \in V$, there exists a path in G going from u to v. A connected acyclic spanning subgraph of G is called a *spanning tree* of G. Let $G - v$ denote the graph obtained by removing from G the node v and its incident edges. A graph G is *2-node connected* if for any $v \in V$, $G - v$ is connected.

A path between two nodes r and s is *shortest* in G if it has minimum length among all the paths in G between r and s. In this case, we denote the path by $P_G(r, s)$, and its length, also known as the *distance* in G between r and s, by $d_G(r, s)$. For a distinguished node $r \in V$, called the *source*, and all the nodes $v \in V \setminus \{r\}$, a *single-source shortest paths tree* (SPT) $S_G(r)$ in G is a spanning tree of G rooted in r and formed by the union of shortest paths, with one shortest path from r to v for each $v \in V \setminus \{r\}$.

Let $P_{G-v}(r, s)$ be a shortest path between r and s in $G - v$, named a *replacement shortest path* for v, and let $d_{G-v}(r, s)$ denote its length. The *most vital node (MVN) problem* with respect to $P_G(r, s)$ asks for finding a node $v^* \in V \setminus \{r, s\}$ such that $d_{G-v^*}(r, s) \geq d_{G-v}(r, s)$, for any $v \in V \setminus \{r, s\}$.

3 An Efficient Solution of the MVN Problem

Let $P_G(r, s) = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path between $r = v_0$ and $s = v_k$ in G. First of all, notice that a node (other than r and s) whose removal increases the distance between r and s must belong to the node set $\{v_1, \ldots, v_{k-1}\}$. Therefore, in the following, we will consider only the removal of the nodes along the path.

3.1 The Structural Properties of Replacement Shortest Paths

In this section, we present the structural properties of replacement shortest paths that will form the basis for the efficiency of our algorithm.

Let $S_G(r)$ denote a SPT in G rooted at r and containing $P_G(r, s)$, and let $v_i, 1 \leq i \leq k - 1$, be a node on $P_G(r, s)$. When node v_i and its incident edges are removed from G, $S_G(r)$ is partitioned into a set of subtrees, that we classify as follows (see Figure 1):

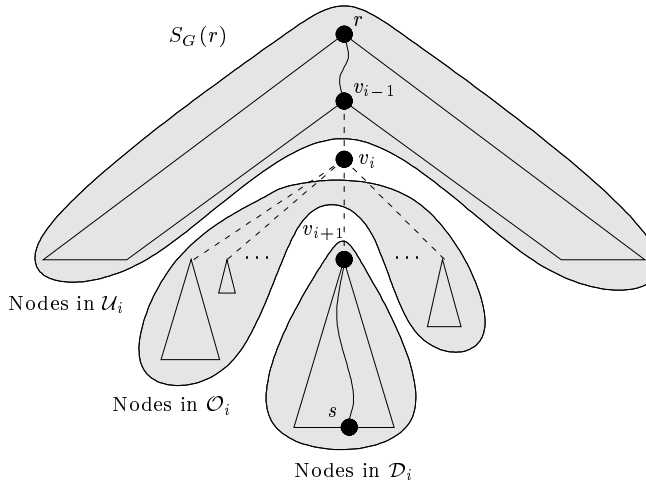


Fig. 1. Node v_i is removed from G: $S_G(r)$ is partitioned into a set of subtrees, with node sets $\mathcal{U}_i$, $\mathcal{O}_i$ and $\mathcal{D}_i$.

1. the subtree of $S_G(r)$ containing the parent v_{i-1} of v_i; we call the nodes of this subtree the *upwards nodes* of v_i, and we denote them as $\mathcal{U}_i$;
2. the subtree of $S_G(r)$ containing the child v_{i+1} of v_i; we call the nodes of this subtree the *downwards nodes* of v_i, and we denote them as $\mathcal{D}_i$;
3. all the remaining subtrees of $S_G(r)$; we call the nodes of the union of all these subtrees the *outwards nodes* of v_i, and we denote them as $\mathcal{O}_i$.

In the rest of the paper, we will make use of the following properties, that hold for $i, j = 1, \ldots, k-1$ and $j \neq i$:

(P1:) $\mathcal{U}_i \cup \mathcal{O}_i \cup \mathcal{D}_i = V \setminus \{v_i\}$;
(P2:) $\mathcal{U}_i, \mathcal{O}_i$ and $\mathcal{D}_i$ are pairwise disjoint;
(P3:) $\mathcal{U}_i \subset \mathcal{U}_{i+1}$;
(P4:) $\mathcal{D}_{i+1} \subset \mathcal{D}_i$;
(P5:) $\mathcal{O}_i \cap \mathcal{O}_j = \emptyset$.

We start by observing that for nodes in $\mathcal{U}_i$, the shortest path to r does not contain v_i. This immediately implies the following:

Lemma 1. *For each node $u \in \mathcal{U}_i$, $d_G(r,u) = d_{G-v_i}(r,u)$.* □

On the other hand, for nodes in $\mathcal{D}_i$, we have that the distance to s does not change when v_i is removed:

Lemma 2. *For each node $u \in \mathcal{D}_i$, $d_G(s,u) = d_{G-v_i}(s,u)$.*

Proof. Suppose, for the sake of contradiction, that $d_G(s,u) \neq d_{G-v_i}(s,u)$. Let $S_G(s)$ be a SPT in G rooted in s containing $P_G(r,s)$. From $d_G(s,u) \neq d_{G-v_i}(s,u)$,

it follows that every shortest path in G between s and u, in particular $P_G(s,u)$ in $S_G(s)$, contains v_i. Therefore, $P_G(s,u)$ has to contain its parent v_{i+1} in $S_G(s)$. Hence, the edge $e_i = (v_i, v_{i+1})$ belongs to $P_G(s,u)$, and since subpaths of shortest paths are shortest paths, we have that

$$d_G(v_{i+1}, u) = w(e_i) + d_G(v_i, u) > d_G(v_i, u).$$

On the other hand, by the fact that $u \in \mathcal{D}_i$, we have that $P_G(r,u)$ in $S_G(r)$ contains v_i and v_{i+1}. Hence, since subpaths of shortest paths are shortest paths, we have that

$$d_G(v_i, u) = w(e_i) + d_G(v_{i+1}, u) > d_G(v_{i+1}, u),$$

that is, we have a contradiction. □

Let $E(\mathcal{U}_i \cup \mathcal{O}_i, \mathcal{D}_i) \subset E$ be the *cut* induced by $\mathcal{D}_i$ in $G - v_i$, i.e., the set of edges of $G - v_i$ with one end node in $\mathcal{U}_i \cup \mathcal{O}_i$ and the other one in $\mathcal{D}_i$. In the following, an edge $f = (x,y) \in E(\mathcal{U}_i \cup \mathcal{O}_i, \mathcal{D}_i)$ will be considered as having node y in $\mathcal{D}_i$. Edges in $E(\mathcal{U}_i \cup \mathcal{O}_i, \mathcal{D}_i)$ will be named the *crossing edges* associated with v_i. Since any replacement shortest path for node v_i must use a crossing edge, the length of the path can be expressed as follows:

$$d_{G-v_i}(r,s) = \min_{f=(x,y) \in E(\mathcal{U}_i \cup \mathcal{O}_i, \mathcal{D}_i)} \{d_{G-v_i}(r,x) + w(f) + d_{G-v_i}(y,s)\}. \quad (1)$$

This immediately suggests an algorithm to solve the MVN problem, but unfortunately we do not know how to compute all the $d_{G-v_i}(r,x)$ distances sufficiently fast, since this might require the computation, for each node v_i on $P_G(r,s)$, of the SPT rooted in r in $G - v_i$. Therefore, let us look at replacement shortest paths more closely. In a path in $G - v_i$ from r to s, let us call the path node $y \in \mathcal{D}_i$ closest to r the *entry node* (into $\mathcal{D}_i$) of the path. We can prove that, to minimize (1), not all the distances in $G - v_i$ between r and nodes in $\mathcal{U}_i \cup \mathcal{O}_i$ need to be computed:

Lemma 3. *Any replacement shortest path for node v_i can be expressed as a concatenation of $P_{G-v_i-\mathcal{D}_i}(r,x)$, edge (x,y) and $P_G(y,s)$, where y is the entry node into $\mathcal{D}_i$, $P_{G-v_i-\mathcal{D}_i}(r,x)$ is a shortest path from r to x in $G - v_i - \mathcal{D}_i$, and $P_G(y,s)$ is a shortest path from y to s in G.*

Proof. Since r is not contained in $\mathcal{D}_i$, but s is, there is a first node on the path traced from r towards s that belongs to $\mathcal{D}_i$. Call that node y, and call its predecessor on the path x. This proves the first and second part of the claim. Part three is due to Lemma 2. □

3.2 Computing Components of Distances

Lemma 3 allows us to compute replacement shortest paths as follows. First, we compute in G a SPT rooted in s. This gives us all distances $d_{G-v_i}(y,s)$ for

$y \in \mathcal{D}_i, i = 1, \ldots, k-1$. Second, we compute all paths $P_{G-v_i-\mathcal{D}_i}(r, x)$ from r to $x \in \mathcal{U}_i \cup \mathcal{O}_i, i = 1, \ldots, k-1$. This is described in more detail in the following paragraph. Third, we compose these distance components by inspecting all crossing edges as we go along the nodes $v_i, i = 1, \ldots, k-1$. This is described in more detail in Section 3.3.

For $x \in \mathcal{U}_i$, from Lemma 1 we have $d_{G-v_i-\mathcal{D}_i}(r, x) = d_G(r, x)$, and therefore the SPT rooted in r that contains $P_G(r, s)$, denoted as $S_G(r)$, gives us all these values. The remaining more interesting task is the computation of $d_{G-v_i-\mathcal{D}_i}(r, x)$ for $x \in \mathcal{O}_i$. We propose to do this as follows, making use of $S_G(r)$. When node $v_i, 1 \leq i \leq k-1$, is removed, we consider the subtree of $S_G(r)$ induced by $\mathcal{U}_i$ – which is of course a SPT rooted in r of the subgraph of G induced by the node set $\mathcal{U}_i$. Then we compute the distance from r to all the nodes in $\mathcal{O}_i$ in the subgraph of G induced by $\mathcal{U}_i \cup \mathcal{O}_i$. We do this by applying Dijkstra's algorithm [4] in the following way to the nodes in $\mathcal{O}_i$, starting from the precomputed distances for $\mathcal{U}_i$. Let $E(\mathcal{U}_i, \mathcal{O}_i)$ be the subset of edges in E having one end node in $\mathcal{U}_i$ and the other one in $\mathcal{O}_i$, let $E(\mathcal{U}_i, x)$ be the subset of edges in $E(\mathcal{U}_i, \mathcal{O}_i)$ having one end node in $\mathcal{U}_i$ and the other one in $x \in \mathcal{O}_i$, and let $E(\mathcal{O}_i, \mathcal{O}_i)$ be the subset of edges in E having both end nodes in $\mathcal{O}_i$. We create an initially empty heap $\mathcal{H}_i$, inserting into it all the nodes $x \in \mathcal{O}_i$, with key

$$k(x) = \begin{cases} \min\limits_{f=(u,x)\in E(\mathcal{U}_i,x)} \{d_{G-v_i}(r, u) + w(f)\} & \text{if } E(\mathcal{U}_i, x) \neq \emptyset; \\ +\infty & \text{otherwise.} \end{cases} \tag{2}$$

Afterwards, we extract the minimum $k(x)$ from $\mathcal{H}_i$; this gives us $d_{G-v_i-\mathcal{D}_i}(r,x)$. Then, we update the keys of adjacent nodes still appearing in $\mathcal{H}_i$, by making use of edges in $E(\mathcal{O}_i, \mathcal{O}_i)$, exactly as in Dijkstra's algorithm. The algorithm iterates until $\mathcal{H}_i$ is empty.

This algorithm has an efficient implementation, as expressed in the following lemma:

Lemma 4. *The values $d_{G-v_i-\mathcal{D}_i}(r, x)$ for all nodes $x \in \mathcal{O}_i, i = 1, \ldots, k-1$, can be computed in $O(m + n \log n)$ time and $O(m)$ space.*

Proof. The initial computation of $S_G(r)$ takes $O(m + n \log n)$ time and $O(m)$ space [5]. Throughout the $k-1$ iterations in our algorithm, $k-1 = O(n)$ heaps are created. Let n_i denote the number of nodes of $\mathcal{O}_i$, and let $m_i = |E(\mathcal{U}_i, \mathcal{O}_i) \cup E(\mathcal{O}_i, \mathcal{O}_i)|$. On heap $\mathcal{H}_i$, we perform $O(n_i)$ *Insert* operations, and from Lemma 1, key initialization can be performed in $O(m_i)$ time once $S_G(r)$ has been computed. Moreover, we perform $O(n_i)$ *ExtractMin* and $O(m_i)$ *DecreaseKey* operations. By using Fibonacci heaps [5], for all the nodes $x \in \mathcal{O}_i$, $d_{G-v_i-\mathcal{D}_i}(r, x)$ can then be computed in $O(m_i + n_i \log n_i)$ time.

Since each *DecreaseKey* operation is associated with an edge of G, and each edge of G is considered at most twice, and given that sets $\mathcal{O}_i$ are disjoint, we finally have that the total time is

$$\sum_{i=1}^{k-1} O(m_i + n_i \log n_i) = O(m + n \log n).$$

□

3.3 Combining Components of Distances

We are now ready to combine the distance components computed so far. We first give a description of the algorithm, and we then analyze its correctness and its time and space complexity.

We consider the nodes $v_1, \ldots, v_{k-1}$ in this order along $P_G(r,s)$, and when the node v_i is considered, we maintain in a heap $\mathcal{H}$ the set of nodes $\mathcal{D}_i$ associated with it. For each node $y \in \mathcal{D}_i$, we consider the subset of edges in $E(\mathcal{U}_i \cup \mathcal{O}_i, \mathcal{D}_i)$ incident to y, denoted as $E(\mathcal{U}_i \cup \mathcal{O}_i, y)$. In the heap, with node y a key $k(y)$ is associated that satisfies the following condition immediately before a *FindMin* operation on $\mathcal{H}$ is performed:

$$k(y) = \min_{f=(x,y)\in E(\mathcal{U}_i \cup \mathcal{O}_i, y)} \{d_{G-v_i-\mathcal{D}_i}(r,x) + w(f) + d_G(y,s)\}. \tag{3}$$

Notice that in general, this key value is not the length of a shortest path in $G-v_i$ from r to s through y, but, as we explained in Section 3.1, we cannot afford to maintain these latter values. We will show later that these keys, however, give us sufficient information to solve the problem.

The algorithm works in stages. At the beginning, the heap $\mathcal{H}$ is created for $\mathcal{D}_0$, that is, all the nodes in the subtree $\mathcal{D}_0$ rooted at v_1 in $S_G(r)$ are inserted, with arbitrarily large keys associated. At the ith stage, we consider the node v_i on $P_G(r,s)$, and we update the heap in the following way:

`Step 1:` We remove from $\mathcal{H}$ the node v_i and the nodes $\mathcal{O}_i$ associated with it. (`Comment:` This leaves exactly the nodes in $\mathcal{D}_i$ in $\mathcal{H}$; we update their keys in Steps 2 and 3.)

`Step 2:` We consider all the nodes in $\mathcal{O}_i$; for each such node x, we inspect its incident edges, and we limit further actions on those crossing into $\mathcal{D}_i$. Let $f = (x,y)$ be one of these crossing edges, if any, and let

$$k' = d_{G-v_i-\mathcal{D}_i}(r,x) + w(f) + d_G(y,s), \tag{4}$$

where $d_{G-v_i-\mathcal{D}_i}(r,x)$ has been computed by means of the procedure described in Section 3.2. If $k' < k(y)$, we decrease the key of y in $\mathcal{H}$ to value k'. (`Comment:` When this step is completed, all the crossing edges associated with v_i and induced by its removal have been exhausted.)

`Step 3:` We then consider all the nodes in $v_{i-1} \cup \mathcal{O}_{i-1}$; for each node x in this set, we look at its incident edges, and we limit further actions on those crossing into $\mathcal{D}_i$. Let $f = (x,y)$ be one of these crossing edges, if any, and let

$$k' = d_G(r,x) + w(f) + d_G(y,s). \tag{5}$$

If $k' < k(y)$, we decrease the key of y in $\mathcal{H}$ to value k'. (`Comment:` When this step is completed, all the crossing edges associated with v_i and induced by the reinsertion of v_{i-1} have been exhausted, and the corresponding key maintenance in the heap is complete.)

`Step 4:` We finally find the minimum of $\mathcal{H}$. (`Comment:` We will prove shortly that the key associated with this minimum is exactly the length of a replacement shortest path in $G - v_i$ between r and s, that is $d_{G-v_i}(r,s)$.)

When all stages $1, \ldots, k-1$ have been completed, a most vital node can then be determined as a node v_j on $P_G(r,s)$ such that

$$d_{G-v_j}(r,s) = \max_{i=1,\ldots,k-1}\{d_{G-v_i}(r,s)\}. \tag{6}$$

Let us now prove that our algorithm indeed computes at each stage the length of a corresponding replacement shortest path.

Lemma 5. *The minimum key found in $\mathcal{H}$ at the ith stage is the length of a replacement shortest path between r and s in $G - v_i$.*

Proof. We prove the lemma in two steps. First, we prove that each key in the heap $\mathcal{H}$, say $k(y)$ for node y, is the length of a shortest path in $G - v_i$ from r to s through the entry node y. The reason is that our algorithm inspects all crossing edges (x,y) incident to y, and keeps track of the best.

Second, we prove that at least one node in the heap has a key corresponding to the length of a replacement shortest path between r and s in $G - v_i$. In fact, for any replacement shortest path P in $G - v_i$, the corresponding entry node y is in $\mathcal{H}$. Let x be its predecessor on P. Then, $k(y)$ equals the length of P, because the prefix of P from r to x is contained in $G - v_i - \mathcal{D}_i$, and then $d_{G-v_i-\mathcal{D}_i}(r,x) = d_{G-v_i}(r,x)$. Therefore, (1) and (3) are both minimized when edge (x,y) is considered, and $k(y) = d_{G-v_i}(r,s)$. □

The following theorem can finally be proved:

Theorem 1. *A most vital node on a shortest path $P_G(r,s)$ between two nodes r and s in a 2-node connected, undirected graph $G = (V,E)$ with n vertices and m edges, with positive real edge lengths, can be determined in $O(m + n\log n)$ time and $O(m)$ space.*

Proof. The correctness of the above algorithm derives from Lemma 5. The time complexity follows from that of Lemma 4 for the initial phase. This allows us to compute (4) in $O(1)$ time for each crossing edge. Clearly, (5) can be computed in $O(1)$ time for each crossing edge as well, once $S_G(r)$ and $S_G(s)$ have been computed. Globally, we perform $O(m)$ computations of (4) and (5), since a crossing edge is checked at most twice, once each in `Step 2` and `Step 3`. Then, we make use of a Fibonacci heap [5] for maintaining $\mathcal{H}$. Since each node of G is inserted into the heap and removed from it at most once, we have a single *MakeHeap*, $O(n)$ *Insert*, $k-1 = O(n)$ *FindMin*, $O(n)$ *Delete* and $O(m)$ *DecreaseKey* operations (since a key may be decreased only when a new crossing edge is considered), and thus we obtain a total time of $O(m + n\log n)$ for heap operations. The time complexity for other tasks is respectively $O(m + n\log n)$ time for computing $S_G(r)$ and $S_G(s)$, $O(n)$ time for managing sets $\mathcal{O}_i, i = 1, \ldots, k-1$, and $O(n)$ time for computing (6). Finally, $O(m)$ space is trivially enough to handle all the operations. Thus, the claim follows. □

4 Discussion

In this paper we have presented a fast solution to the problem of finding a most vital node along a shortest path $P_G(r, s)$ between two nodes r and s in a graph G. Implicitly, our algorithm computes not only the lengths of replacement paths, but also the paths themselves; it can be easily modified to do so explicitly. It runs in $O(m + n \log n)$ time and $O(m)$ space, which, as far as we know, is the first improvement over the trivial bound of $O(nm + n^2 \log n)$ time and $O(m)$ space that we get by recomputing a replacement shortest path between r and s from scratch after the removal of each node along $P_G(r, s)$.

In some applications, such as transportation networks, it appears to be more realistic to associate costs with both, nodes and edges, instead of only one type of network components. Our approach also answers the corresponding more general question that suggests itself: In a graph where both edges and nodes have a non-negative cost, and where both edges and nodes can fail, what is a most vital edge or node on a shortest path? The algorithm can be modified slightly and still runs within the same asymptotic bounds for this more general question, for two reasons. First, edge failures can be modelled as node failures, when each edge is replaced by a path of length two with an extra node in the center of that path; then, the failure of the extra node represents the failure of the original edge. Second, Dijkstra's algorithm can be adapted easily to work also for shortest path computations in graphs with costs on edges and nodes, where the cost of a path is the sum of the costs of its edges and nodes. Obviously, both modifications do not change the asymptotic bounds for the runtime and the storage space.

Our algorithmic solution is also useful in quite a different application context. In large networks, components (nodes and edges) may be owned by different owners. The incentive of an owner of a component to forward a message, naturally, is to get some reward. In standard economic terms, that reward is the price of the service of forwarding the message. It is economically desirable that each owner declares the true price for the service that its component offers, so as to allocate the overall resources in a best possible way. Nevertheless, there is an incentive for owners to speculate and ask for a higher price, in the hope of getting a higher profit. This leads to economically suboptimal resource allocation and is therefore undesirable. A few studies in the computer science literature have devoted their attention to setting the boundary conditions in such a way that speculating with high prices does not pay off. This is known as *mechanism design* for selfish agents [9,10,11]. In [10], Nisan is explicitly suggesting a rewarding model for forwarding messages on paths, based on microeconomic theory, that requires the computation of replacement path lengths for edges. This model assumes that only edges charge a price for forwarding a message; nodes perform their service for free. Here, again, it would be more realistic to have a price for both, edges and nodes, than a limitation of the pricing to the edges alone. A straightforward modification of the charging scheme from [10] to node and edge prices serves the desired purpose. Now, the modification of our algorithm for node and edge costs and failures is an efficient implementation of the required replacement path computations.

Our solution is efficient, but it is still open whether it is optimal. Notice that to improve our solution, a faster computation of a single source shortest paths tree must be provided. For more general settings there are still many open problems; one of them deals with multiple edge or node failures on a shortest path.

References

1. M.O. Ball, B.L. Golden and R.V. Vohra, Finding the most vital arcs in a network, *Oper. Res. Letters*, **8** (1989) 73–76.
2. A. Bar-Noy, S. Khuller and B. Schieber, The complexity of finding most vital arcs and nodes. TR CS-TR-3539, Institute for Advanced Studies, University of Maryland, College Park, MD, 1995.
3. H.W. Corley and D.Y. Sha, Most vital links and nodes in weighted networks, *Oper. Res. Letters*, **1** (1982) 157–160.
4. E.W. Dijkstra, A note on two problems in connection with graphs, *Numer. Math.*, **1** (1959) 269–271.
5. M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the ACM*, **34** (3) (1987) 596-615.
6. K. Malik, A.K. Mittal and S.K. Gupta, The k most vital arcs in the shortest path problem, *Oper. Res. Letters*, **8** (1989) 223–227.
7. E. Nardelli, G. Proietti and P. Widmayer, Finding the detour-critical edge of a shortest path between two nodes, *Info. Proc. Letters*, **67** (1) (1998) 51–54.
8. E. Nardelli, G. Proietti and P. Widmayer, A faster computation of the most vital edge of a shortest path between two nodes, *Info. Proc. Letters*, to appear. Also available as TR 15-99, Dipartimento di Matematica Pura ed Applicata, University of L'Aquila, L'Aquila, Italy, April 1999.
9. N. Nisan, Algorithms for selfish agents, *Proc. of the 16th Symp. on Theoretical Aspects of Computer Science (STACS'99)*, Lecture Notes in Computer Science, Vol. 1563, Springer, (1999) 1–15.
10. N. Nisan and A. Ronen, Algorithmic mechanism design, *Proc. of the 31st Annual ACM Symposium on Theory of Computing (STOC'99)*, (1999) 129–140.
11. J.S. Rosenschein and G. Zlotkin, *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge, Massachusetts, 1994.
12. R.E. Tarjan, Efficiency of a good but not linear set union algorithm, *Journal of the ACM*, **22** (1975) 215–225.
13. S. Venema, H. Shen and F. Suraweera, A parallel algorithm for the single most vital vertex problem with respect to single source shortest paths, *Online Proc. of the First Int. Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT'2000)*, Chapter 22, `http://www2.comp.polyu.edu.hk/PDCAT2000/publish.html`.

Algorithm for the Cost Edge-Coloring of Trees

Xiao Zhou and Takao Nishizeki

Graduate School of Information Sciences, Tohoku University
Aoba-yama 05, Sendai, 980-8579, Japan.
{zhou, nishi}@ecei.tohoku.ac.jp

Abstract. Let C be a set of colors, and let ω be a cost function which assigns a real number $\omega(c)$ to each color c in C. An edge-coloring of a graph G is to color all the edges of G so that any two adjacent edges are colored with different colors. In this paper we give an efficient algorithm to find an optimal edge-coloring of a given tree T, that is, an edge-coloring f of T such that the sum of costs $\omega(f(e))$ of colors $f(e)$ assigned to all edges e is minimum among all edge-colorings of T. The algorithm takes time $O(n\Delta^2)$ if n is the number of vertices and Δ is the maximum degree of T.

1 Introduction

A *vertex-coloring of a graph* $G = (V, E)$ is to color all the vertices of G so that any two adjacent vertices are colored with different colors. We denote by n the number of vertices in G. Let $C = \{c_1, c_2, \cdots, c_m\}$ be a set of colors, and assume that $m = |C|$ is sufficiently large, say $m = n$. Let $\omega : C \to \mathbf{R}$ be a cost function which assigns a real number $\omega(c) \in \mathbf{R}$ to each color $c \in C$. The *cost vertex-coloring problem* is to find a vertex-coloring $f : V \to C$ of G such that $\sum_{v \in V} \omega(f(v))$ is as small as possible. One can observe that the cost vertex-coloring problem is NP-hard since the ordinary vertex-coloring problem is NP-hard. The cost vertex-coloring problem remains NP-hard for interval graphs, but the problem can be solved in linear time for trees [9]. The "vertex-chromatic sum problem" in [10,11] is merely the cost vertex-coloring problem for a special case where $\omega(c_i) = i$ for each index $i \geq 1$. Jansen gives a polynomial-time algorithm to solve the vertex-chromatic sum problem for graphs of bounded treewidth [5].

In this paper we introduce the cost edge-coloring problem. An *edge-coloring* $f : E \to C$ *of a graph* $G = (V, E)$ is to color all the edges of G so that any two adjacent edges are colored with different colors. The minimum number of colors required for edge-colorings of G is called the *chromatic index*, and is denoted by $\chi'(G)$. An ordinary edge-coloring problem is to compute the chromatic index $\chi'(G)$ of a given graph G and find an edge-coloring of G with $\chi'(G)$ colors. On the other hand, the *cost edge-coloring problem* is to find an *optimal edge-coloring* of G, that is, an edge-coloring f such that $\sum_{e \in E} \omega(f(e))$ is minimum among all edge-colorings of G. An optimal edge-coloring does not always use the minimum number $\chi'(G)$ of colors. For example, suppose that $\omega(c_1) = 1$ and $\omega(c_i) = 5$ for each index $i \geq 2$, then the graph G with $\chi'(G) = 3$ in Figure 1(a) can be

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 288–297, 2001.
© Springer-Verlag Berlin Heidelberg 2001

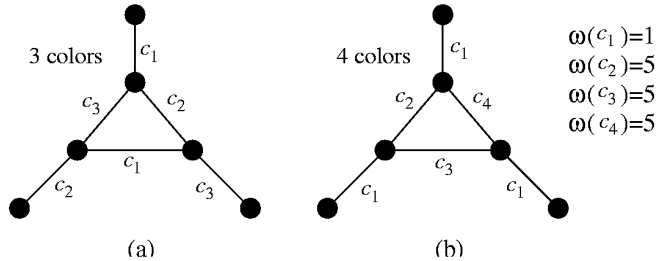


Fig. 1. (a) Edge-coloring with $\chi'(G)$ colors and (b) optimal edge-coloring with $\chi'(G) + 1$ colors.

uniquely colored with the three cheapest colors c_1, c_2 and c_3 as in Figure 1(a), but the edge-coloring is not optimal; an optimal edge-coloring of G uses the four cheapest colors c_1, c_2, c_3 and c_4 as depicted in Figure 1(b). The *edge-chromatic sum problem* introduced by Giaro and Kubale [4] is merely the cost edge-coloring problem for a special case where $\omega(c_i) = i$ for each index $i \geq 1$. The cost edge-coloring problem has a natural application in scheduling theory. Consider the scheduling of unit execution time of biprocessor tasks on dedicated machines. An example of such tasks is the file transfer problem in a computer network in which each file engages two corresponding nodes, sender and receiver, simultaneously [3]. Another example is the biprocessor diagnostic problem in which links execute concurrently the same test for a fault tolerant multiprocessor system [6]. These problems can be modeled by a graph G in which machines correspond to the vertices and tasks correspond to the edges. An edge-coloring of G corresponds to a schedule, where the edges colored with color $c_i \in C$ represent the collection of tasks that are executed in the ith time slot. For each i, if a task is executed in the ith time slot, then it takes the cost $\omega(c_i)$. The goal is to find a schedule that minimizes the total cost. This corresponds to the cost edge-coloring problem.

One can observe that the cost edge-coloring problem is NP-hard since the ordinary edge-coloring problem is NP-hard. Furthermore, the cost edge-coloring problem remains NP-hard for bipartite graph, because the edge-chromatic sum problem is NP-hard for bipartite graphs [4]. When restricted to trees, many NP-hard problems can be efficiently solved in polynomial time [1,2,13,14]. Giaro and Kubale recently give an algorithm to solve the edge-chromatic sum problem for trees T in time $O(n\Delta^{3.5} \log n)$, where n is the number of vertices and Δ is the maximum degree of T [4]. However, it is not known whether the cost edge-coloring problem can be solved in polynomial time for trees.

In this paper we give an algorithm to solve the cost edge-coloring problem for trees T in time $O(n\Delta^2)$. The algorithm hence runs in linear time if Δ is a fixed constant. The algorithm takes time $O(n\Delta^{1.5} \log(\Delta N))$ if all the color costs $\omega(c)$ are integers in the range $[-N, N]$. The edge-chromatic sum problem is a special case of the cost edge-coloring problem in which $N = n$. Our algorithm can therefore solve the edge-chromatic sum problem for trees

in time $O(n\Delta^{1.5}\log n)$. The time-complexity $O(n\Delta^{1.5}\log n)$ is better than the time-complexity $O(n\Delta^{3.5}\log n)$ of Giaro and Kubale's algorithm [4]. We also show that any tree T has an optimal edge-coloring using exactly $\chi'(T) = \Delta(T)$ colors.

2 Preliminaries

In this section we define some terms and present easy observations. Let $T = (V, E)$ denote a tree with vertex set V and edge set E. We will use notions such as *root, child, descendant* and *leaf* in their usual meaning. We denote by n the number of vertices in T. T is a "free tree," but we choose for the sake of convenience an arbitrary vertex as a root r, and regard T as a "rooted tree." The *degree* $deg(v)$ of vertex v is the number of edges incident to v. We denote the maximum degree of T by $\Delta(T)$ or simply by Δ. We denote by $d(v)$ the number of children of v in T. Then $d(v) = deg(v)$ if $v = r$, and $d(v) = deg(v) - 1$ otherwise. An edge joining vertices u and v is denoted by (u, v). For a vertex v of T, the subtree of T induced by v and all descendants of v is called *the subtree of* T *rooted at* v, and is denoted by T_v. Clearly, $T = T_r$ for the root r.

Since tree T has $n-1$ edges, one may assume without loss of generality that the color set C satisfies $|C| = n - 1$. We however assume for the notational sake of convenience that $|C| = n$, and write $C = \{c_1, c_2, \cdots, c_n\}$. An *edge-coloring* $f : E \to C$ of a tree T is to color all edges of T by colors in C so that any two adjacent edges are colored with different colors. Let $\omega : C \to \mathbf{R}$, where $\mathbf{R}$ is the set of real numbers. One may assume with loss of generality that ω is non-decreasing, that is, $\omega(c_i) \leq \omega(c_{i+1})$ for any index i, $1 \leq i < n$. The *cost* $\omega(f)$ *of an edge-coloring* f of a tree $T = (V, E)$ is defined as follows:

$$\omega(f) = \sum_{e \in E} \omega(f(e)).$$

An edge-coloring f of T is called an *optimal* one if $\omega(f)$ is minimum among all edge-colorings of T. The *cost edge-coloring problem* is to find an optimal edge-coloring of a given tree T. The cost of an optimal edge-coloring of tree T is called the *minimum cost of tree* T, and is denoted by $\omega(T)$.

Let f be an edge-coloring of T. For each vertex v of T, let $C(f, v)$ be the set of all colors that are assigned to edges incident to v, that is,

$$C(f, v) = \{f(e) \mid e \text{ is an edge incident to } v \text{ in } T\}.$$

We say that a color $c \in C$ is *missing at* v if $c \notin C(f, v)$. Let $\mathrm{Miss}(f, v)$ be the set of all colors missing at v, that is,

$$\mathrm{Miss}(f, v) = C - C(f, v).$$

Interchanging colors in an alternating path is one of the standard techniques of an ordinary edge-coloring, which we also use in the paper. Let f be an edge-coloring of a tree $T = (V, E)$, let c_i and c_j be any two colors in C, and let

$T(c_i, c_j)$ be the subgraph of T induced by all the edges colored with c_i and c_j. Since T is a tree, each connected component of $T(c_i, c_j)$ is a path of length one or more, whose edges are colored alternately with c_i and c_j. We call such a path a *c_ic_j-alternating path*. A vertex $v \in V$ is an end of a c_ic_j-alternating path if and only if exactly one of c_i and c_j is missing at v. If v is an end of a c_ic_j-alternating path, then the path is denoted by $P(v; c_i, c_j)$. The set of all edges in $P(v; c_i, c_j)$ is sometimes simply denoted by $P(v; c_i, c_j)$. Interchanging colors c_i and c_j in $P(v; c_i, c_j)$, one can obtain another edge-coloring f' of T. We then have the following lemma on f'.

Lemma 1. *If $c_i \in \mathrm{Miss}(f, v)$, $c_j \in C(f, v)$, and $i < j$, then $\omega(f') \leq \omega(f)$, $c_i \in C(f', v)$, $c_j \in \mathrm{Miss}(f', v)$, and $C(f', v) \cup \{c_j\} = C(f, v) \cup \{c_i\}$.*

Proof. The colors of all edges assigned by f are the same as those by f' except for the edges in $P(v; c_i, c_j)$. The first edge in $P(v; c_i, c_j)$ is colored with c_j by f, but it is colored with c_i by f'. We therefore have $c_i \in C(f', v)$, $c_j \in \mathrm{Miss}(f', v)$, and $C(f', v) \cup \{c_j\} = C(f, v) \cup \{c_i\}$. Since $\omega(c_i) \leq \omega(c_j)$ and $P(v; c_i, c_j)$ starts with an edge colored with c_j, we have

$$\sum_{e \in P(v; c_i, c_j)} \omega(f'(e)) \leq \sum_{e \in P(v; c_i, c_j)} \omega(f(e)).$$

We thus have $\omega(f') \leq \omega(f)$. $\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

3 Algorithm

In this section we prove the following theorem.

Theorem 1. *An optimal edge-coloring of a tree T can be found in time $O(n\Delta^2)$ if T has n vertices and Δ is the maximum degree of T.*

In the remainder of this setion we prove Theorem 1. In Section 3.1 we give an algorithm to compute the minimum cost $\omega(T)$ of a given tree T, and in Section 3.2 we give an algorithm to find an optimal edge-coloring of T.

3.1 Computing the Minimum Cost $\omega(T)$

A "dynamic programming method" is a standard one to solve a combinatorial problem on trees. We also use it, and compute the minimum cost $\omega(T)$ of a given tree T by the "bottom-up tree computation." However, one can observe that the minimum cost $\omega(T_v)$ of a subtree T_v rooted at a vertex v cannot be computed directly from the minimum costs $\omega(T_{v_j})$, $1 \leq j \leq d(v)$, of all the children v_j of v. (See Figure 2.)

Our first idea is to introduce a new parameter $\omega(T_v, i)$ defined for each vertex v of T and each color $c_i \in C$ as follows:

$$\omega(T_v, i) = \min\{\omega(f) \mid \ f \text{ is an edge-coloring of } T_v \text{ and } c_i \in \mathrm{Miss}(f, v)\}.$$

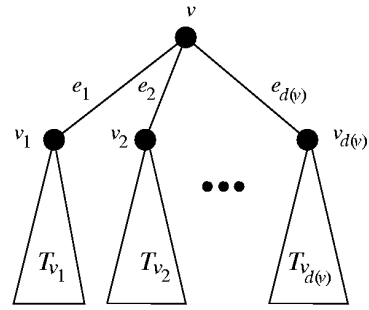


Fig. 2. Subtree T_v rooted at v.

Since T has exactly $n-1$ edges and $|C| = n$, T_v has an edge-coloring f in which c_i is not used by f and hence $c_i \in \mathrm{Miss}(f, v)$. Thus $\omega(T_v, i)$ is well-defined. Clearly,

$$\omega(T_v) = \min_{1 \le i \le n} \omega(T_v, i).$$

We compute the values $\omega(T_v, i)$ for all indices i, $1 \le i \le n$, from leaves to root r. Thus the DP table for each vertex v consists of the n values $\omega(T_v, i)$, $1 \le i \le n$, and has a size n.

The second idea is to reduce the size from n to $d(v) + 1$. We first have the following two lemmas on the properties of $\omega(T_v, i)$.

Lemma 2. *For any vertex v of tree T, $\omega(T_v, i)$ is non-increasing with i, that is, $\omega(T_v, i) \ge \omega(T_v, i+1)$ for each index i, $1 \le i \le n-1$.*

Proof. Let f be an edge-coloring of T_v such that $c_i \in \mathrm{Miss}(f, v)$ and $\omega(f) = \omega(T_v, i)$. If $c_{i+1} \in \mathrm{Miss}(f, v)$, then the definition of $\omega(T_v, i+1)$ implies that $\omega(T_v, i) = \omega(f) \ge \omega(T_v, i+1)$. One may thus assume that $c_{i+1} \in C(f, v)$. Then, interchanging colors c_i and c_{i+1} in $P(v; c_i, c_{i+1})$, we obtain another edge-coloring f' of T. By Lemma 1 $\omega(f) \ge \omega(f')$ and $c_{i+1} \in \mathrm{Miss}(f', v)$. Therefore by the definition we have

$$\omega(T_v, i) = \omega(f) \ge \omega(f') \ge \omega(T_v, i+1).$$

$\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

Lemma 3. *For any vertex v of tree T,*

$$\omega(T_v, d(v)+1) = \omega(T_v, d(v)+2) = \cdots = \omega(T_v, n).$$

Proof. Let i be any index such that $d(v) + 2 \le i \le n$. Then by Lemma 2 $\omega(T_v, d(v)+1) \ge \omega(T_v, i)$. It therefore suffices to prove that $\omega(T_v, d(v)+1) \le \omega(T_v, i)$.

Let f be any edge-coloring of T_v such that $c_i \in \mathrm{Miss}(f, v)$ and $\omega(f) = \omega(T_v, i)$. Since $|C(f, v)| = d(v)$, at least one of the $d(v)+1$ colors $c_1, c_2, \cdots, c_{d(v)+1}$ is missing at v. Let c_j be any of them, then $1 \le j \le d(v)+1$ and $c_j \in \mathrm{Miss}(f, v)$. By the definition of $\omega(T_v, j)$ we have

$$\omega(T_v, j) \le \omega(f) = \omega(T_v, i). \tag{1}$$

Since $j \le d(v) + 1$, by Lemma 2

$$\omega(T_v, d(v) + 1) \le \omega(T_v, j). \tag{2}$$

Thus by Eqs. (1) and (2) we have $\omega(T_v, d(v) + 1) \le \omega(T_v, i)$. $\mathcal{Q.E.D.}$

From Lemmas 2 and 3 we immediately have the following lemma.

Lemma 4. *For any vertex v, $\omega(T_v) = \omega(T_v, d(v) + 1)$. In particular, $\omega(T) = \omega(T_r, d(r) + 1)$.*

Thus our DP table for each vertex v consists of only the $d(v) + 1$ values $\omega(T_v, i)$, $1 \le i \le d(v) + 1$, and hence the size is $d(v) + 1$. All the values $\omega(T_v, i)$ for i, $d(v) + 2 \le i \le n$, are equal to $\omega(T_v, d(v) + 1)$.

For each leaf v of T, the DP table for v consists of a single value $\omega(T_v, 1) = 0$. Note that $d(v) = 0$ and T_v has no edge.

Thus it suffices to show how to construct the DP table for an internal vertex v from the DP tables for all the children $v_1, v_2, \cdots, v_{d(v)}$ of v. (See Figure 2.) We first notice that, for each i, $1 \le i \le d(v)+1$, T_v has an edge-coloring f such that $\omega(f) = \omega(T_v, i)$, $c_i \in \mathrm{Miss}(f, v)$, and $C(f, v)$ consists of the first $d(v)$ colors in C other than c_i, as in the following lemma whose proof is omitted in this extended abstract.

Lemma 5. *For any vertex v of T and any index i, $1 \le i \le d(v) + 1$, T_v has an edge-coloring f such that $\omega(f) = \omega(T_v, i)$, $c_i \in \mathrm{Miss}(f, v)$, and $C(f, v) = D_i$, where $D_i = \{c_k \mid 1 \le k \le d(v) + 1, k \ne i\}$.*

We then compute the value $\omega(T_v, i)$ for an index i, $1 \le i \le d(v)+1$, from the $d(v) \times (d(v) + 1)$ values $\omega(T_{v_j}, k)$ for all indices j, $1 \le j \le d(v)$, and all indices k, $1 \le k \le d(v) + 1$. Let J be the set of indices j of all children v_j of v, that is, $J = \{j \mid 1 \le j \le d(v)\}$. Let $e_j = (v, v_j)$ for each $j \in J$, as illustrated in Figure 2. Let $1 \le i \le d(v) + 1$, and let K_i be the set of indices k of all colors c_k in D_i, that is,

$$K_i = \{k \mid 1 \le k \le d(v) + 1, k \ne i\}.$$

Suppose that f is an edge-coloring of T_v such that $\omega(f) = \omega(T_v, i)$, $c_i \in \mathrm{Miss}(f, v)$, and $C(f, v) = D_i$. Lemma 5 implies that T_v has such an edge-coloring f. For each $j \in J$, let f_j be the restriction of f to the edges in T_{v_j}, that is, f_j is an edge-coloring of T_{v_j} such that $f_j(e) = f(e)$ for every edge e in T_{v_j}. Then clearly

$$\omega(f) = \omega(T_v, i) = \sum_{j \in J} \omega(f_j) + \omega(D_i), \tag{3}$$

where

$$\omega(D_i) = \sum_{c \in D_i} \omega(c).$$

Note that the value $\omega(D_i)$ depends on i, but does not depend on the edge-coloring f. For each $j \in J$, we denote by j' the index such that $c_{j'} = f(e_j) \in D_i$. Since f is an edge-coloring of T_v, all colors $f(e_j)$ are different from each other. Therefore the mapping $\varphi : J \to K_i$ such that $\varphi(j) = j' \in K_i$ for each $j \in J$ is a bijection. Since $c_{j'} \in \mathrm{Miss}(f_j, v_j)$, the definition of $\omega(T_{v_j}, j')$ implies that $\omega(f_j) \geq \omega(T_{v_j}, j')$. However, we have

$$\omega(f_j) = \omega(T_{v_j}, j') \tag{4}$$

as follows: if $\omega(f_j) > \omega(T_{v_j}, j')$, then T_{v_j} has an edge-coloring f'_j such that $c_{j'} \in \mathrm{Miss}(f'_j, v_j)$ and $\omega(f'_j) = \omega(T_{v_j}, j') < \omega(f_j)$, and hence the edge-coloring f' of T_v obtained from f by replacing f_j with f'_j satisfies $c_i \in \mathrm{Miss}(f', v)$ and $C(f', v) = D_i$ but $\omega(f') < \omega(f) = \omega(T_v, i)$, a contradiction. By Eqs. (3) and (4) we have

$$\omega(f) = \omega(T_v, i) = \sum_{j \in J} \omega(T_{v_j}, j') + \omega(D_i). \tag{5}$$

Suppose conversely that $\varphi : J \to K_i$ is any bijection. For each $j \in J$, we denote $\varphi(j)$ by j', that is, $j' = \varphi(j)$, and let g_j be any edge-coloring of T_{v_j} such that $\omega(g_j) = \omega(T_{v_j}, j')$ and $c_{j'} \in \mathrm{Miss}(g_j, v_j)$. Let g be an edge-coloring of T_v extended from all g_j, $j \in J$, as follows:

$$g(e) = \begin{cases} c_{j'} & \text{if } e = (v, v_j) \text{ and } j \in J; \\ g_j(e) & \text{if } e \text{ is an edge of } T_{v_j} \text{ and } j \in J. \end{cases}$$

Then $c_i \in \mathrm{Miss}(g, v)$, $C(g, v) = D_i$, and

$$\begin{aligned} \omega(g) &= \sum_{j \in J} \omega(g_j) + \omega(D_i) \\ &= \sum_{j \in J} \omega(T_{v_j}, j') + \omega(D_i). \end{aligned} \tag{6}$$

Let

$$b(i) = \min_{\varphi} \sum_{j \in J} \omega(T_{v_j}, j'), \tag{7}$$

where the minimum is taken over all bijections $\varphi : J \to K_i$. Then one can know from Eqs. (5) and (6) that $\omega(g) = \omega(T_v, i)$ if and only if φ attains the minimum value $b(i)$.

We thus have the following lemma.

Lemma 6. *For any internal vertex v of T and any index i, $1 \leq i \leq d(v) + 1$,*

$$\omega(T_v, i) = b(i) + \omega(D_i).$$

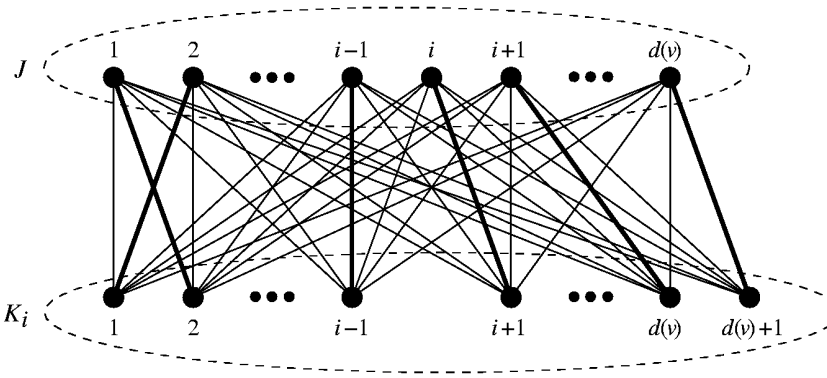


Fig. 3. Complete bipartite graph $B(i) = (J, K_i; J \times K_i)$.

One can easily compute the values $\omega(D_i)$, $1 \leq i \leq d(v)+1$, in time $O(d(v))$. We shall therefore show how to compute $b(i)$.

The problem of computing $b(i)$ can be reduced to the minimum weight perfect matching problem for a complete bipartite graph, as follows. Let $B(i) = (J, K_i; J \times K_i)$ be a complete bipartite graph with vertex sets J and K_i. (See Figure 3.) Note that $|J| = |K_i| = d(v)$. Let $h : J \times K_i \to \mathbf{R}$ be a weight function such that $h(e) = \omega(T_{v_j}, k)$ for each edge $e = (j, k)$, $j \in J$ and $k \in K_i$. The minimum weight perfect matching problem is to find a perfect matching M such that the sum of weights of the edges in M is as small as possible [12]. A perfect matching in $B(i)$ is indicated by thick lines in Figure 3. Clearly, $b(i)$ is equal to the weight of the minimum perfect matching in $B(i)$.

The minimum weight perfect matching problem can be solved in time $O(d^3(v))$ for a single graph $B(i)$ [12], and hence the value $b(i)$ for an index i, $1 \leq i \leq d(v)+1$, can be computed in time $O(d^3(v))$. Therefore the $d(v)+1$ values $b(i)$ for all indices i, $1 \leq i \leq d(v)+1$, can be computed total in time $O(d^4(v))$.

The third idea is to compute all the $d(v)+1$ values $b(i)$, $1 \leq i \leq d(v)+1$, together. Kao *et al.* recently showed that if a given complete bipartite graph B has d vertices then one can solve in time $O(d^3)$ the minimum weight perfect matching problems for all d complete bipartite graphs obtained from B by deleting each of the d vertices [7,8]. Let $K = \{1, 2, \cdots, d(v)+1\}$, let $B = (J, K; J \times K)$ be a complete bipartite graph, and let $h(e) = \omega(T_{v_j}, k)$ for each edge $e = (j, k)$, $j \in J$ and $k \in K$. Applying their algorithm to B, we can compute all the values $b(i)$, $1 \leq i \leq d(v)+1$, in time $O(d^3(v))$. We thus have the following lemma.

Lemma 7. *For any internal vertex v of T, one can construct the DP table for v from the DP tables for all the children of v in time $O(d^3(v))$.*

We can thus compute the minimum cost $\omega(T) = \omega(T_r, d(r)+1)$ in the DP table for root r in time

$$O\left(\sum_{v\in V} d^3(v)\right) = O\left(\left(\sum_{v\in V} d(v)\right)\Delta^2\right)$$
$$= O(n\Delta^2).$$

Clearly, the total size of all DP tables is $\sum_{v\in V}(d(v)+1) = O(n)$. We do not store the bijection φ found at each internal vertex.

3.2 Finding an Optimal Edge-Coloring

We next show how to find an optimal edge-coloring f of T. Although we compute the DP tables from leaves to the root, we decide the colors of edges from the root to leaves, as follows.

When we computed

$$\omega(T) = \omega(T_r, d(r)+1) = b(d(r)+1) + \omega(D_{d(r)+1}),$$

we found a bijection $\varphi : J \to K_{d(r)+1}$ attaining the minimum value $b(d(r)+1)$ in Eq. (7), where $J = K_{d(r)+1} = \{j \mid 1 \le j \le d(r)\}$. We find the bijection φ again. For each j, $1 \le j \le d(r)$, we color the edge e_j joining r and its jth child r_j with $c_{\varphi(j)} \in D_{d(r)+1} = \{c_1, c_2, \cdots, c_{d(r)}\}$. We hence have $C(f, r) = \{c_1, c_2, \cdots, c_{deg(r)}\}$ since $d(r) = deg(r)$.

We may thus assume that, for an internal vertex v, the edge joining v and its parent u has been colored with a color $c_i \in C$ for some index i, $1 \le i \le deg(u)$. Let $i' = i$ if $i \le d(v)+1$, and let $i' = d(v)+1$ if $i \ge d(v)+2$. We find again the bijection $\varphi : J \to K_{i'}$ attaining the minimum value $b(i')$ in Eq. (7). For each j, $1 \le j \le d(v)$, we color the edge $e_j = (v, v_j)$ with $c_{\varphi(j)} \in D_{i'}$. We hence have $C(f, v) = D_{i'} \cup \{c_i\} \subseteq \{c_1, c_2, \cdots, c_{deg(v)}\} \cup \{c_i\}$ since $d(v)+1 = deg(v)$.

Thus we can correctly find an optimal edge-coloring of T. Clearly, the algorithm takes time $O(n\Delta^2)$. This completes a proof of Theorem 1.

Kao *et al.* also show that if a given complete bipartite graph B has d vertices and all edge weights are integers in the range $[-N, N]$ then one can solve in time $O(d^{2.5}\log(dN))$ the minimum weight perfect matching problems for all d complete bipartite graphs obtained from B by deleting each of the d vertices [7,8]. Using their algorithm, one can find an optimal edge-coloring of tree T in time $O(n\Delta^{1.5}\log(\Delta N))$ if all the color costs are integers in the range $[-N, N]$.

We finally give a remark. As we mentioned in Introduction, an optimal edge-coloring of a graph G does not always use the minimum number $\chi'(G)$ of colors. However, from the algorithm we immediately have the following theorem for trees T. Note that $\chi'(T) = \Delta(T)$ and that our algorithm does not use the method of interchanging colors in an alternating path although it is used in the proofs of Lemmas 2 and 5.

Theorem 2. *Any tree T has an optimal edge-coloring using the first $\Delta(T)$ cheapest colors $c_1, c_2, \cdots, c_\Delta$ in C.*

Acknowledgments

We thank Tak-Wah Lam and Takeshi Tokuyama for pointing out some references.

References

1. S. Arnborg and J. Lagergren. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.
2. R.B. Borie, R.G. Parker, and C.A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.
3. E.G. Coffman, M.R. Garey, D.S. Johnson, and A.S. LaPaugh. Scheduling file transfers. *SIAM J. Comput.*, 14:744–780, 1985.
4. K. Giaro and M. Kubale. Edge-chromatic sum of trees and bounded cyclicity graphs. *Information Processing Letters*, 75:65–69, 2000.
5. K. Jansen. The optimum cost chromatic partition problem. In *Proc. CIAC'97, Lecture Notes in Computer Science, Springer-Verlag*, 1203:25–36, 1997.
6. H. Krawczyk and M. Kubale. An approximation algorithm for diagnostic test scheduling in multicomputer systems. *IEEE Trans. Comput.*, 34:869–872, 1985.
7. M.Y. Kao, T.W. Lam, W.K. Sung and H.F. Ting. All-cavity maximum matchings. In *Proc. ISAAC'97, Lecture Notes in Computer Science, Springer-Verlag*, 1350:364–373, 1997.
8. M.Y. Kao, T.W. Lam, W.K. Sung and H.F. Ting. Cavity matchings, label compressions, and evolutionary trees. *SIAM J. Comp.*, 30(2):602–624, 2000.
9. L.G. Kroon, A. Sen, H. Deng, and A. Roy. The optimal cost chromatic partition problem for trees and interval graphs. In *Proc. WG'96 International Workshop on Graph Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Springer*, 1197:279–292, 1997.
10. M. Kubale. *Introduction to Computational Complexity and Algorithmic Graph Coloring.* Gdańskie Towarzystwo Naukowe, Gdańsk, Poland, 1998.
11. E. Kubicka. *The Chromatic Sum of a Graph.* Ph.D. Thesis, Western Michigan University, 1989.
12. C.H. Padadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, NJ, 1982.
13. J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Discrete Math.*, 10:529–550, 1997.
14. X. Zhou, S. Nakano, and T. Nishizeki. Edge-coloring partial k-trees. *Journal of Algorithms*, 21:598–617, 1996.

Counting H-Colorings of Partial k-Trees*

Josep Díaz, Maria Serna, and Dimitrios M. Thilikos

Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya, Campus Nord – Mòdul C5,
c/Jordi Girona Salgado, 1-3. E-08034, Barcelona, Spain
{diaz,mjserna,sedthilk}@lsi.upc.es

Abstract. The problem of counting all H-colorings of a graph G of n vertices is considered. While the problem is, in general, #P-complete, we give linear time algorithms that solve the main variants of this problem when the input graph G is a k-tree or, in the case where G is directed, when the underlying graph of G is a k-tree. Our algorithms remain polynomial even in the case where $k = O(\log n)$ or in the case where the size of H is $O(n)$. Our results are easy to implement and imply the existence of polynomial time algorithms for a series of problems on partial k-trees such as core checking and chromatic polynomial computation.

1 Introduction

In this paper we consider a series of counting problems associated with various versions of homomorphisms from a graph G to a fixed graph H. Given a graph $H = (V(H), E(V))$, with loops but without multiple edges, for any graph $G = (V(G), E(G))$ an *homomorphism* of G to H is a map $\theta : V(G) \to V(H)$ with the property that $\{v, w\} \in E(G) \Rightarrow \{\theta(v), \theta(w)\} \in E(H)$. We will call such a homomorphism a *H-coloring* of G. For H fixed, the *H-coloring problem* asks if there is an H-coloring of G. If H is bipartite or it has a loop, then the H-coloring problem can be trivially solved in polynomial time, but in the case that H is not-bipartite and loopless the problem is known to be NP-complete [HN90] (see also [GHN00] for the complexity of the same problem for bounded degree graphs).

An earlier version of the H-coloring problem is the *exact H-coloring problem*: given a graph G, decide if there is an H-coloring θ of G such that $\theta(V(G)) = V(H)$ and $\theta(E(G)) = E(H)$. This problem appeared in [GJ79] as problem GT-52 and is known to be NP-complete, even for the case where H is a triangle [Lev73]. We also consider the intermediate problem of asking whether there is an H-coloring θ of G such that $\theta(V(G)) = V(H)$ and we call it *vertex exact H-coloring*.

Given a graph G, let $\mathcal{H}(G, H)$ denote the set of all H-colorings of G, and let $\mathcal{E}(G, H)$ denote the set of all exact H-colorings of G. Also, we denote as

* Research supported by the EU project ALCOM-FT (IST-99-14186). The research of the 3rd author was supported by the Ministry of Education and Culture of Spain, Grant number MEC-DGES SB98 0K148809.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 298–307, 2001.
© Springer-Verlag Berlin Heidelberg 2001

$\mathcal{V}(G,H)$ the set of all vertex H-colorings of G. For a fixed graph H, denote by *#H-coloring* the problem of, given G as input, computing $|\mathcal{H}(G,H)|$. Also, denote by *#EH-coloring* the problem of, given as input G, computing $|\mathcal{E}(G,H)|$ and by *#VEH-coloring* the problem of, given as input G, computing $|\mathcal{V}(G,H)|$. Dyer and Greehill [DG00] have proved that unless every connected component of H is an isolated vertex without a loop, a complete graph with all loops present or a complete unlooped bipartite graph, the $\#H$-coloring problem is #P-complete, even when G has bounded degree. It is easy to see that the same result hold for the $\#EH$-coloring ($\#VEH$-coloring) as the knowledge of $\mathcal{E}(G,H')$ ($\mathcal{V}(G,H')$) for any (induced) subgraph H' of H implies the knowledge of $\mathcal{H}(G,H)$. For further negative results on the approximability of the $\#H$-coloring see [CDF] and [DFJ99]. For positive and negative complexity results concerning special cases of the $\#H$-coloring problem, depending on restrictions on both H and G, see [Edw86, DNS00].

The notion of *treewidth* appears to play a central role in many areas of graph theory. Roughly, a graph has small treewidth if it can be constructed by assembling small graphs together in a tree structure, namely a tree decomposition of small width (see section 2 for the formal definitions). It was first defined by Robertson and Seymour and served as one of the cornerstones of their lengthy proof of the Wagner conjecture, known now as the Graph Minors Theorem (for a survey see [RS85]). Treewidth appears to have interesting applications in algorithmic graph theory. In particular, a wide range of, otherwise intractable, combinatorial problems are polynomially, even linearly, solvable when restricted to graphs with bounded treewidth (alternatively, graphs with bounded treewidth are called *partial k-trees*). Several general results have been developed for proving, constructively, the existence of such algorithms. Among them, we mention the work of Courcelle in [Cou90b] where the existence of polynomial algorithms is guarantied by the expressability of the corresponding problem by Monadic Second Order formulas (see also [Cou90a]). Similar results have been developed in [CM93] for solving counting problems for partial k-trees (see also [CMR]). As a consequence of these results, it is possible to contruct a polynomial time algorithm solving the $\#H$-coloring problem for partial k-trees when k and the size of H are fixed constants. Unfortunately, the general results of Courcelle, do not provide implementable algorithms because of the immense hidden constants in their complexity. Alternatively, the attention of many researchers was fixed on developing fast and easy to implement "tailor made" algorithms for problems of specific interest. The standard methodology to get such solutions consists of a two step procedure: First to find a tree-decomposition of the input graph, with treewidth bounded by a constant, although possibly not optimal, and second to use some kind of *dynamic programming* taking advantage of the bounded treewidth decomposition of the graph, to get a solution (for a survey, see [Bod97]). In particular, such a dynamic programing technique, solving – among others – the H-coloring problem for partial k-trees, has been developed in [TP97] (also, see [JS97], for counting list colorings and other generalizations of colorings). For the first canonical step, Bodlaender in [Bod96] presented, for

any k, a linear time algorithm that, given a graph G, checks whether G is a partial k-tree and, if so, outputs an mimimum width tree decomposition. As this algorithm appears extremely hard to be implemented (see also [BK96]), the first canonical step can be carried out by one of the previously developoed algorithms that are able to output a tree-decomposition of G with treewidth bounded by a constant (linear in k) [Ree92]. In particular, the deterministic algorithm by Reed [Ree92] runs in $O(n \log n)$ time, which, for constant k, gives a 2 approximation to the optimal treewidth-decomposition. Moreover, its implementation is easier than the algorithm in [Bod96]. Consequently, we will assume that any partial k-tree is always accompanied with its linear size tree decomposition of bounded width.

In this paper we present a polynomial time algorithm for the counting problems $\#H$-coloring, $\#EH$-coloring, and $\#VEH$-coloring, in the case that the input graph G has constant treewidth. Although our methodology follows the two cannonical steps, we shall remark that we present an easy to be implemented algorithm for the second step, asumming the aforementioned results on the existence of fast and implementable algorithms for obtaining a bounded treewidth-decomposition for a partial k-tree G. Finally, we stress out that our results are the first "tailor made" positive results on counting H-colorings for the case where H is generic and G is restricted.

In Section 2 we review the basic definitions and some preliminary results on treewith. In Section 3 we present the algorithm solving the $\#H$-coloring problem. In Section 4 we present two variants of the algorithm of Section 3, for solving the $\#VEH$-coloring and the $\#EH$-coloring problem respectively. In the final section, we present some consequences of our previous results for a series of problems on graphs with bounded treewidth.

2 Definitions and Basic Results

Robertson and Seymour introduced the notion of treewidth in the framework of the theory of graph minors.

A *tree decomposition* of a graph G is a pair (X, U) where U is a tree whose vertices we will call *nodes* and $X = (\{X_i \mid i \in V(U)\})$ is a collection of subsets of $V(G)$ such that

1. $\bigcup_{i \in V(U)} X_i = V(G)$,
2. for each edge $\{v, w\} \in E(G)$, there is an $i \in V(U)$ such that $v, w \in X_i$, and
3. for each $v \in V(G)$ the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of U.

The *width* of a tree decomposition $(\{X_i \mid i \in V(U)\}, U)$ equals $\max_{i \in V(U)}\{|X_i| - 1\}$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. Computing treewidth is an NP-complete problem [ACP93].

We now define some extensions of the definition of a tree decomposition were introduced by Boadlaender and Kloks [Bod96] (see also [Arn85]). A *rooted* tree decomposition is a triple $D = (X, U, r)$ in which U is a tree rooted at r and (X, U) is a tree decomposition. Let $D = (X, U, r)$ be a rooted tree decomposition

of a graph G where $X = \{X_i \mid i \in V(U)\}$. D is called a *nice* tree decomposition if the following are satisfied:

1. Every node of U has at most two children,
2. if a node i has two children j, h then $X_i = X_j = X_h$,
3. if a node i has one child, then either $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ or $|X_i| = |X_j| - 1$ and $X_i \subset X_j$.

The following lemma can be found in [BK96].

Lemma 1. *For any constant $k \geq 1$, given a tree decomposition of a graph G of width $\leq k$ and $O(n)$ nodes, where n is the number of vertices in G, there exists an $O(n)$ algorithm that constructs a nice tree decomposition of G of width less or equal than k and with at most $O(n)$ nodes.*

We observe that a nice tree decomposition $D = (\{X_p \mid p \in V(U)\}, U, r)$ contains nodes of the following four possible types. A node $p \in V(U)$ is called: *Start*, if p is a leaf, *Join*, if p has two children $q_i, i = 1, 2$, *Forget*, if p has only one child q and $|X_p| = |X_q| - 1$, *Introduce*, if p has only one child q and $|X_p| = |X_q| + 1$.

Notice that for every start node we may assume that $|X_p| = 1$: the effect of *start* nodes with $|X_p| > 2$ can be obtained by using a *start* node with a set containing one vertex, and then $|X_p| - 1$ *introduce* nodes, which add all the other vertices. For the purposes of this paper we will assume that the root r is a *forget* node with $X_r = \emptyset$. Taking in mind Lemma 1, we will assume that any partial k-tree is given along with a nice tree decomposition of it.

Let $D = (X, U, r)$ be a nice tree decomposition of a graph G. For each node p of D, let U_p be the subtree of U, rooted at node p. We set $V_p = \cup_{v \in V(U_p)} X_v$ and, for any $p \in V(U)$, we define $G_p = G[V_p]$. Notice that $G_r = G$.

3 An Algorithm for Counting the Number of H-Colorings

Let H be a fixed graph. Given an input graph G together with a nice tree decomposition $D = (X, U, r)$ of it, we wish to count $|\mathcal{H}(G, H)|$, i.e. to solve the $\#H$-coloring problem for G.

The algorithm will do a *traversal* of D computing and tabulating for each $p \in V(U)$ the appropriate information. Before describing it, we need some additional definitions.

Given a nice tree decomposition $D = (X, U, r)$ of a graph G, for each $p \in V(U)$, let $F_p = \{\varphi : X_p \to V(H)\}$. Notice that if G has treewidth k, then for any $p \in V(U)$, $|F_p| \leq h^{k+1}$. Moreover, for the root we have $F_r = \{\varnothing\}$.

The table associated to a node $p \in V(U)$ will have an entry for each $\varphi \in F_p$, holding the value $I_p(\varphi) = |\{\theta \in \mathcal{H}(G_p, H) \mid \theta|_{X_p} = \varphi\}|$ (in general, if θ is a function, we denote $\theta|_S = \{(v, a) \in \theta \mid v \in S\}$). As we always have $\theta|_\emptyset = \varnothing$, we get $|\mathcal{H}(G, H)| = I_r(\varnothing)$.

The function count-H takes as input G, D and a topological ordering $S = (p_1, \ldots, p_s)$ of $V(U)$ and outputs the value $I_r(\varnothing)$.

```
function count-H(G, H, D, S)

    D = (X, U, r) is a nice tree decomposition of G with width k
    n = |V(G)|, h = |V(H)|, s = |V(U)|
    S = (p_1, ..., p_s) is a topological ordering of V(U)
    begin
        for p := p_1 to p_s do
            if p is a start node, with X_p = {v} then
                for all a ∈ V(H) do I_p((v, a)) := 1
            end
            if p is a introduce node then
                let q be its unique child
                v := X_p − X_q
                S_q := {u ∈ X_q | {u, v} ∈ E(G_p)}
                for all φ ∈ F_q and a ∈ V(H) do
                    if ∀_{u∈S_q}{φ(u), a} ∈ E(H) then
                        I_p(φ ∪ {(v, a)}) := I_q(φ)
                    else I_p(φ ∪ {(v, a)}) := 0
                    end
                end for
                erase the information on node q
            end
            if  is a forget node then
                let q be its unique child
                v := X_q − X_p
                for all φ ∈ F_p do
                    I_p(φ) := Σ_{a∈V(H)} I_q(φ ∪ {(v, a)})
                end for
                erase the information on node q
            end
            if p is a join node with children q_1, and q_2 then
                for all φ ∈ F_p do
                    I_p(φ) := I_{q_1}(φ) · I_{q_2}(φ)
                end for
                erase the information on nodes q_1 and q_2
            end
        end for
        return I_r(∅)
    end
```

Theorem 1. *Given a graph G with $n = |V(G)|$, a nice tree decomposition $D = (X, U, r)$ of G with width k, and a topological ordering of the nodes in D. Then there exists an algorithm that computes the number of H-colorings of G in $O(nh^{k+1} \min\{k, h\})$ steps using $O(h^{k+1} \log n)$ additional space, where $h = |V(H)|$.*

Notice that for constant treewidth k, we can allow $h = O(n)$ and still have a polynomial time algorithm. Taking in mind the algorithm of Bodlaender in [Bod96] we have the following.

Corollary 1. *Given a graph G if it has treewidth bounded by a constant k, then we can compute $|\mathcal{H}(G, H)|$ in $O(nh^{k+1} \min\{k, h\}) + O(f(k)n)$ steps.*

For constant k and h, our algorithm works in linear time and therefore we have the following.

Corollary 2. *The #H-coloring problem, with parameters the treewidth of the input and the size of H, is a fixed parameter tractable problem (i.e. belongs to the class FPT).*

By suitably altering function count-H we can have the analogous of Theorem 1 and Corrolary 1 for the $\#VEH$-coloring and the $\#EH$-coloring problems.

Theorem 2. *Given a graph G with $n = |V(G)|$, a nice tree decomposition $D = (X, U, r)$ of G with width k, and a topological ordering of the nodes in D. Then there exists an algorithm that computes the number of vertex exact H-colorings of G in $O(nh^{k+1}2^h \max\{k, 2^{2h}\})$ time using $O(h^{k+1}2^h \log n)$ additional space, where $h = |V(H)|$.*

Corollary 3. *Given a graph G with n vertices and treewidth bounded by a constant k, then we can compute $|\mathcal{V}(G, H)|$ in $O(nh^{k+1}2^h \max\{k, 2^{2h}\}) + O(f(k)n)$ time, where $h = |V(H)|$.*

Theorem 3. *Given a graph G with $n = |V(G)|$, a nice tree decomposition $D = (X, U, r)$ of G with width k, and a topological ordering of the nodes in D. Then there exists an algorithm that computes the number of H-colorings of G in $O(nh^{k+1}2^{h+e} \max\{k, 2^{2(h+e)}\})$ time using $O(h^{k+1}2^{h+e} \log n)$ additional space, where $h = |V(H)|$ and $e = |E(H)|$.*

Corollary 4. *Given a graph G with n vertices and treewidth bounded by a constant k, then we can compute $|\mathcal{E}(G, H)|$ in $O(nh^{k+1}2^{h+e} \max\{k, 2^{2(h+e)}\} + O(f(k)n)$ time, where $h = |V(H)|$ and $e = |E(H)|$.*

Notice that for each of the algorithms given so far, if we retain the information of all the tables, it is possible to use it in order to enumerate all the homomorphisms. The storing of all the tables implies a burden of $O(n/\log n)$ to the space reported in Theorems 1, 2, and 3. In particular, setting up a suitable bookkeeping of the enumerated homomorphisms, a top-down traversal of the table information can pop-up each of them in $O(n)$ steps.

We point out that in all algorithms of this section and for constant h, we can allow $k = O(\log n)$ and still have polynomial time. Finally, all the results of this section can be adapted straighforwardly to solve the directed versions of the problems (H and G are directed graphs).

4 Extensions to Other Results

In this section, we present some consequences of the polynomial time algorithms given in the previous sections. For each problem, either we give a new polynomial-time result or we improve previously known results. Recall that if we are given a graph G, directed or undirected, all of our algorithms should be used in conjuction with one of the algorithms mentioned in the introduction, that in polynomial time, compute the treewidth decomposition of bounded width.

4.1 Coloring Problems

Notice that a graph G is H-colorable if $|\mathcal{H}(G,H)| \geq 1$. This implies the following corollary of Theorems 1, 2, and 3 for the corresponding decision version of the problems examined.

Corollary 5. *For any graph H with h vertices and e edges, we can solve, for an n-vertex partial k-tree as input graph,*

- *the H-coloring problem in $O(nh^{k+1}\min\{k,h\})$ steps,*
- *the vertex exact H-coloring problem in $O(nh^{k+1}2^h\max\{k,2^h\})$ steps, and*
- *the exact H-coloring problem in $O(nh^{k+1}2^{h+e}\max\{k,2^{h+e}\})$ steps.*

Corollary 5 improves the time of the best known algorithm for the H-coloring of partial k-trees. This algorithm runs in $O(nh^{2(k+1)})$ steps and is due to Telle and Proskurowski in [TP97].

We denote as K_c the complete graph with c vertices. As the problem of counting the number of c-colorings of a graph G is equivalent to the $\#K_c$-coloring problem we can conclude to the following.

Corollary 6. *An algorithm can be constructed to compute the number of c-colorings of a partial k-tree of n vertices in $O(nc^{k+1}\min\{k,c\})$ steps.*

We denote as $\beta_G(c)$ the function mapping c to the number of colorings of G that use exactly c colors, i.e. $\beta_G(c) = |\mathcal{V}(G,K_c)|$.

Lemma 2. *For any graph G, $\beta_G(c) = |\mathcal{H}(G,K_c)| - \sum_{1\leq r\leq c-1} \binom{c}{r}\beta_G(r)$.*

The formula of Lemma 2 and Corollary 6 give a way to compute $\beta_G(c)$.

Lemma 3. *An algorithm can be constructed that, given partial k-tree G with n vertices and for any $c, 1 \leq c \leq n$, computes $\beta_G(c)$ in $O(nc^{k+2}\min\{k,c\})$ steps.*

Corollary 7. *An algorithm can be constructed that computes the chromatic polynomial of a partial k-tree with n vertices in $O(kn^{k+3})$ steps.*

In our knowledge, the best algorithm for computing the chromatic polymonial for partial k-trees can be derived from the algorithm of Andrzejak [And98]. This algorithm can compute in $O(n^{2+7\log_2 c})$ steps the Tutte polynomial of a partial k-tree of n vertices, where c is twice the number of partitions of a set with $3(k+1)$ elements. Several researchers have considered the problem of counting the number of proper c-colorings in a graph G. The problem is #P-hard for $c \geq 3$ and maximum degree $\Delta \geq 3$ [Jer95]. In the same paper, it is proved that there exists a Fully Polynomial time Randomized Aproximation Scheme (FPRAS) for the number of colorings in the case when $c \geq 2\Delta + 1$. Recently, Bubley et all. in [BDGJ99] proved that the problem is #P-hard for fixed Δ, but there is a FPRAS for $c = 5$ and $\Delta = 3$. Edwards [Edw86] proved that if $c \geq 3$ and the minimum degree where $\delta \geq \alpha n$, $n = |V(G)|$, the counting problem is #P-complete if $\alpha < \frac{c-2}{c-1}$, but it is in P for $\alpha > \frac{c-2}{c-1}$.

4.2 Problems on Cores

Given a graph G, a *core* of G is a subgraph C of G such that G is homomorphic to C, but G fails to be homomorphic to any proper subgraph of C. This notion of core is due to Hell and Nešetřil [HN92]. A graph G is *a core* if G is a core for itself or, equivalently, if $|\mathcal{E}(G,G)| = |\mathcal{H}(G,G)|$. For example, any graph C_{2k+1} is a core (C_{2k+1} is the connected 2-regular graph with $2k+1$ vertices). It is known, that in general, the problem of deciding whether G is a core is NP-complete [HN92] and in the same paper it is presented a polynomial time algorithm, to decide if G is a core, for the particular case when G has *independence number* $\alpha(G) \leq 2$. Finally, we mention that "almost all graphs" are cores [Nešˇ99].

We first prove the following.

Corollary 8. *An algorithm can be constructed that decides, in $O(k^2n^{k+3})$ steps, whether a partial k-tree with n vertices is a core.*

Proof. Notice that it is enough to check whether there exists a subgraph H of G where $|E(H)| = |E(G)| - 1$ and such that $|\mathcal{H}(G,H)| \geq 1$. According to Theorem 1, this requires $O(|E(G)|kn^{k+2}) = O(k^2n^{k+3})$ steps (take in mind that if G is a partial k-tree then $E(G) = O(k|V(G)|)$).

We mention that the property of being a core is an EMS-property (i.e. involves counting or summing evaluations over sets definable on monadic second order logic) and therefore, the results in [Cou90b, ALS91] imply the existence of a polynomial time algorithm for deciding it. So far, no explicit algorithm has been reported for this problem.

We consider the isomorphism problem on cores. Notice that graphs G and H are isomorphic iff $|\mathcal{E}(G,H)| \cdot |\mathcal{E}(H,G)| > 1$. This check can be done in polynomial time when both G and H are cores and partial k-trees.

Corollary 9. *An algorithm can be constructed that checks, in $O(kn^{k+2})$ steps, if two cores G and H of treewidth $\leq k$ are isomorphic ($n = \max\{|V(G)|, |V(H)|\}$).*

In general, checking whether two partial k-trees are isomorphic can be done in $O(n^{k+4.5})$ steps [Bod90]. Corollaries 8 and 9 improve this time for the case where some of the input graphs is a core.

Using the idea of Corollary 9, we can count automorphisms of cores with bounded treewidth.

Corollary 10. *An algorithm can be constructed that outputs, in $O(kn^{k+2})$ steps, the number of automorphisms of a core G of n vertices and treewidth $\leq k$.*

A graph G is called *rigid* when the identity is the only homomorphism in $\mathcal{H}(G,G)$ [Nešˇ99]. We conclude to the following.

Corollary 11. *An algorithm can be constructed that checks, in $O(kn^{k+2})$, steps whether a n-vertex partial k-tree is rigid or not.*

4.3 Problems on Counting Independent Sets

In the case where H is the unique graph with two vertices one edge and one loop, $|\mathcal{H}(G,H)|$ is the number of independent sets of G. We have the following.

Corollary 12. *An algorithm can be constructed that outputs the number of independent sets of a n-vertex partial k-tree in $O(n2^{k+1})$ steps.*

Notice that the linear algorithm of Corollary 12 remains polynomial even if the treewidth of G is bounded by $O(\log n)$. We could remark that, by specializing the structure of H, we can generate an arbitrary number of counting problems that are, in general, #P-complete. As an example we mention the problem of counting the q-particle Widom-Rowlinson configurations in bounded treewidth graphs [DG00].

Suppose that H is a star with a loop attached on its unique internal vertex (a star is a tree of diameter 2). Then, asking whether $|\mathcal{V}(G,H)| \geq 1$ is equivalent to ask whether G has an independent set of size $\geq k$. Finally, Theorem 2 yields the following.

Corollary 13. *An algorithm can be constructed that for any partial k-tree of n vertices and any r, outputs the number of the independent sets of G with at least r vertices in $O(n(r+1)^{k+1}2^{r+1}\max\{k, 2^{2(r+1)}\})$ steps.*

Acknowledgments

We wish to thank Rafel Cases for his useful remarks on this research.

References

[ACP93] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Mathematics*, 8:277–284, 1993.

[ALS91] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[And98] Artur Andrzejak. An algorithm for the Tutte polynomials of graphs of bounded treewidth. *Discrete Mathematics*, 190(1-3):39–54, 1998.

[Arn85] Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985.

[BDGJ99] Russ Bubley, Martin Dyer, Catherine Greenhill, and Mark Jerrum. On approximately counting colorings of small degree graphs. *SIAM Journal on Computing*, 29(2):387–400, 1999.

[BK96] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21:358–402, 1996.

[Bod90] Hans Leo Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11:631–643, 1990.

[Bod96] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[Bod97] Hans L. Bodlaender. Treewidth: algorithmic techniques and results. In *Mathematical foundations of computer science 1997 (Bratislava)*, pages 19–36. Springer, Berlin, 1997.

[CDF] Colin Cooper, Martin Dyer, and Alan Frieze. On Marcov chains for randomly H-coloring a graph. *Journal of Algorithms* (to appear).

[CM93] Bruno Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comp. Sc.*, 109:49–82, 1993.

[CMR] B. Courcelle, J.A. Makowski, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic. *Discrete Applied Mathematics* (to appear).

[Cou90a] Bruno Courcelle. Graph rewriting: an algebraic and logical approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B*, pages 192–242, Amsterdam, 1990. North Holland Publ. Comp.

[Cou90b] Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.

[DFJ99] Martin Dyer, Alan Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. In *40th Annual Symposium on Foundations of Computer Science*, pages 210–217, 1999.

[DG00] M.E. Dyer and C. S. Greenhill. The complexity of counting graph homomorphisms. In *11th ACM/SIAM Symposium on Discrete Algorithms*, pages 246–255, 2000.

[DNS00] Josep Díaz, Jaroslav Nešetřil, and Maria Serna. H-coloring of large degree graphs. Technical Report No. 2000-465, KAM-DIMATIA Series, Charles University, 2000.

[Edw86] K. Edwards. The complexity of coloring problems on dense graphs. *Theoretical Computer Science*, 16:337–343, 1986.

[GHN00] Anna Galluccio, Pavol Hell, and Jaroslav Nešetřil. The complexity of H-colouring of bounded degree graphs. *Discrete Math.*, 222(1-3):101–109, 2000.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[HN90] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, series B*, 48:92–110, 1990.

[HN92] P. Hell and J. Nešetřil. The core of a graph. *Discrete Mathematics*, 109:117–126, 1992.

[Jer95] M. Jerrum. A very simple algorithm for stimating the number of k-colorings of a low degree graph. *Random Structures and Algorithms*, 7:157–165, 1995.

[JS97] Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Appl. Math.*, 75(2):135–155, 1997.

[Lev73] L. Levin. Universal sequential search problems. *Problems of Information Transmissions*, 9:265–266, 1973.

[Neš99] Jaroslav Nešetřil. Aspects of structural combinatorics (graph homomorphisms and their use). *Taiwanese J. Math.*, 3(4):381–423, 1999.

[Ree92] B. Reed. Finding approximate separators and computing tree-width quickly. In *24th ACM Symposium on Theory of Computing*, pages 221–228, 1992.

[RS85] Neil Robertson and Paul D. Seymour. Graph minors — a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge Univ. Press, 1985.

[TP97] J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM Journal on Discrete Mathematics*, 10:529–550, 1997.

A Linear Time Algorithm for Enumerating All the Minimum and Minimal Separators of a Chordal Graph

L. Sunil Chandran

Dept. of Computer Science and Automation,
Indian Institute of Science, Bangalore, 560012, India
sunil@csa.iisc.ernet.in

Abstract. We give $O(m+n)$ algorithms for enumerating all *minimum separators* as well as *minimal separators* in a connected chordal graph. We give a tight upper bound, $n-\kappa(G)-1$, for the number of minimal separators in a chordal graph. An important contribution of the paper is our characterisation of minimal (minimum) separators of chordal graphs and the above results are obtained as consequences of the characterisations.

1 Introduction

Let C be a cycle in a graph G. A chord of C is an edge of G joining two vertices of C which are not consecutive. A graph G is called a chordal (or triangulated) graph iff every cycle in G of length 4 or more has a chord. Chordal graphs arise in many applications (see [6,15,20]). Chordal graphs constitute one of the most important subclasses of perfect graphs [6].

In a connected graph G, a *separator* S is a subset of vertices whose removal separates G into at least two connected components. S is called a $(a-b)$ *separator* iff it disconnects vertices a and b. A $(a-b)$ separator is said to be a *minimal separator* iff it does not contain any other $(a-b)$ separator. A *minimum separator* is a separator with minimum size. Clearly every minimum separator is a minimal separator. The *(vertex) connectivity*, $\kappa(G)$ is defined to be the size of a minimum separator. The connectivity of a complete graph is by definition $|V|-1$. It doesn't have any separators. For basic notions about connectivity, we refer to [7,6].

The problem of listing all minimal separators and minimum separators is one of the fundamental enumeration problems in graph theory, which has great practical importance in reliability analysis for networks and operations research for scheduling problems [9,5,1].

The related problem of finding the connectivity of a graph is conjectured to have an $O(m+n)$ algorithm (see [4] for history). The interest in finding better or different algorithms is still very much alive [19,16,17,3,22,18,4]. The latest and the fastest in the sequence is due to Gabow [4], which runs in $O(n^2k + nk(min(k^{\frac{5}{2}}, kn^{\frac{3}{4}})))$ time, where $k=\kappa(G)$. In this paper, we present an $O(m+n)$

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 308–317, 2001.
© Springer-Verlag Berlin Heidelberg 2001

algorithm, restricted to the class of chordal graphs, which not only finds the connectivity, but lists all minimum separators.

The problem of listing all minimum or minimal separators of an undirected graph is considered by various authors [5,9,10,8]. In [9], a $O(2^k n^3)$ algorithm is given for listing all *minimum* separators. A $O(n^6 R_\Sigma)$ algorithm is given is [10], to list all *minimal* separators, where R_Σ is the total number of minimal separators in the graph. This is improved in [8] to $O(n^3 R_\Sigma^+ + n^4 R_\Sigma)$, where $R_\Sigma^+ \leq (n(n-1)/2 - m)R_\Sigma$. Restricted to the class of chordal graphs, we present algorithms that lists all minimal (or minimum) separators in $O(m+n)$ time, which compares sharply with the general case running times available in [9,10,8]. Algorithms to enumerate the minimal separators for some other subclasses of perfect graphs (eg. permutation graphs) are given in [12,11].

As an auxiliary result we show that the number of minimal separators in a chordal graph is upper bounded by $n - \kappa(G) - 1$. We also show the tightness of this upper bound by constructing a graph on n nodes with number of minimum separators = number of minimal separators = $n - \kappa(G) - 1$. Some other important classes of graphs which are known to have polynomial number of minimal separators include permutation graphs, circular permutation graphs, trapezoidal graphs, circle graphs, circular arc graphs, distance hereditary graphs, chordal bipartite graphs, cocomparability graphs of bounded dimension and weakly triangulated graphs [11,12,13,14]. Note that in general graphs, the number of minimal separators can be exponential [10]. In [9] it is shown that in a general graph, the number of minimum separators is bounded by $O(2^k \frac{n^2}{k})$, where $k = \kappa(G)$. For information on why number of minimal (or minimum) separators are important see [10,12].

An important contribution of this paper is the structural understanding of separators of chordal graphs in general and the minimum and minimal separators in particular (Lemma 5, Theorems 1, 2). As far as we know, these characterisations are new. Also we haven't seen any algorithm for listing the minimal separators of a chordal graph elsewhere. But we point out that an algorithm which runs in worse than linear time (eg $O(n^2(m+n))$) may be designed with weaker understanding of the separator structure. The upper bound (Theorem 5) can be obtained in other ways also, though we haven't seen it mentioned anywhere else.

The only fact about chordal graphs, the reader should be familiar with, for reading this paper, apart from the notion of PEO, (which will be explained in the next subsection) is a well known characterisation due to Dirac [2,6].

Lemma 1. *A graph G is a chordal graph if and only if every minimal separator of G induces a complete subgraph.*

1.1 PEO, Some Definitions, and Notations

Let $G = (V, E)$ be a connected undirected graph. Throughout this paper we will use V for the set of vertices of G and E for the set of edges. $|V|$ and $|E|$ will be denoted by n and m respectively. $N(v)$ will denote the set of neighbours of v,

that is $N(v) = \{u \in V : (u, v) \in E\}$. For $A \subseteq V$, we use $N(A)$ to denote the set $\bigcup_{v \in A} N(v) - A$. The subgraph of G induced by the nodes in A will be denoted by $G(A)$.

A bijection $f : V \to \{1, 2, \cdots, n\}$ is called an ordering of the vertices of G. Then $f(v)$ is referred to as the number associated with the vertex v, or simply the *number of* v with respect to the ordering f. Given an ordering f of a graph G, we define the following terms.

Definition 1. *Let $A \subseteq V$. The highest(A) is defined to be the vertex with the highest number in A. Similarly lowest(A) is the vertex in A with the lowest number.*

Definition 2. *A path $P = (w_1, w_2, \cdots, w_k)$ in G is called an increasing path, iff $f(w_1) < f(w_2) < \cdots < f(w_k)$. It is called a decreasing path iff $f(w_1) > f(w_2) > \cdots > f(w_k)$. A single node can be considered as either increasing or decreasing.*

Definition 3. *A vertex $u \in N(v)$ is called a higher neighbour of v iff $f(u) > f(v)$. The set of higher neighbours of v will be denoted by $N_h(v)$, ie*

$$N_h(v) = \{u \in N(v) : f(u) > f(v)\}$$

Similarly the set of lower neighbours of v is denoted by $N_l(v)$.

$$N_l(v) = \{u \in N(v) : f(u) < f(v)\}$$

$d_h(v) = |N_h(v)|$ and $d_l(v) = |N_l(v)|$.

Definition 4. *An ordering f of G is called a perfect elimination ordering (PEO) iff for each $v \in V$, $G(\{v\} \bigcup N_h(v))$ is a complete subgraph (clique) of G. Then $f(v)$ will be called the PEO number of v.*

Note that every graph may not have a PEO. In fact it is known that a graph G is chordal if and only if there exists a PEO for G [6]. A $O(m + n)$ algorithm to construct a PEO for a chordal graph G is given in [21]. Note that there can be more than one PEO for a given chordal graph. The observations presented in this paper are valid with respect to any PEO. Therefore we just use $PEO(v)$ to denote the PEO number of v. It will be clear from the context which PEO is being referred to.

From now on we are only interested in chordal graphs. We will be using G to denote an arbitrary connected chordal graph. We close this subsection with one more definition.

Definition 5. *A vertex v is called an interior vertex of G, iff there exists a vertex $u \notin N(v)$ with $PEO(u) > PEO(v)$. Note that v is an interior vertex if and only if $d_h(v) < n - PEO(v)$. We denote the set of interior vertices by V'. That is, $V' = \{v \in V : d_h(v) < n - PEO(v)\}$.*

2 The Minimum Separators

A chordless path from u to v is defined to be a path from u to v in G such that no two non consecutive nodes of the path are adjacent. The reader can easily verify that if there is a path between u and v then there is a chordless path also. For example, the shortest path between u and v has to be a chordless path.

Lemma 2. *Let $P = (w_1, w_2, \cdots, w_k)$ be a chordless path in a chordal graph G and let $w_i = highest(P)$. Then $(w_1, w_2, \cdots, w_i)$ is an increasing path while $(w_i, w_{i+1}, \cdots, w_k)$ is a decreasing path.*

Proof: Consider first the path $(w_1, w_2, \cdots, w_i)$. Since $w_i = highest(P)$, this cannot be a decreasing path. If $i \leq 2$, obviously this is an increasing path. Let $i > 2$. If $(w_1, w_2, \cdots, w_i)$ is not an increasing path, then there should be a node $w_j \in P$, $1 < j < i$, such that $PEO(w_j) < PEO(w_{j+1})$ and $PEO(w_j) < PEO(w_{j-1})$. Then $\{w_{j-1}, w_{j+1}\} \subseteq N_h(w_j)$. By definition of PEO, $N_h(w_j)$ forms a clique, and we have $(w_{j-1}, w_{j+1}) \in E$. So (w_{j-1}, w_{j+1}) forms a chord for P, contradicting the chordless property of P. We infer that $(w_1, w_2, \cdots, w_i)$ is an increasing path. A similar argument shows that $(w_i, w_{i+1}, \cdots, w_k)$ should be a decreasing path. □

Corollary 1. *Let P=$(w_1, w_2, \cdots, w_k)$ be a chordless path and w_k=$highest(P)$. Then P is an increasing path.*

Corollary 2. *Let $P = (w_1, w_2, \cdots, w_k)$ be a chordless path and $PEO(w_1) < PEO(w_k)$. Then $PEO(w_2) > PEO(w_1)$.*

Lemma 3. *Let $P = (w_1, w_2, \cdots, w_k)$ be an increasing path in a chordal graph. Let $PEO(u) > PEO(w_k)$ where $u \in N(w_1)$. Then $u \in N_h(w_k)$.*

Proof: We prove the lemma by induction on $|P|$, the number of nodes in P. For $|P| = 1$, the lemma is trivial. Assume that for all increasing paths with $|P| = k-1$, where $k > 1$, the lemma is true. Let $P = (w_1, w_2, \cdots, w_k)$ be an increasing path with $|P| = k$. Note that we have $PEO(u) > PEO(w_k) > PEO(w_{k-1})$ since P is an increasing path. Applying the induction assumption on the $(k-1)-$node path $(w_1, w_2, \cdots, w_{k-1})$, we get $u \in N_h(w_{k-1})$. Now $\{w_k, u\} \subseteq N_h(w_{k-1})$ and by definition of PEO, $N_h(w_{k-1})$ is a clique, and $(u, w_k) \in E$. Remembering $PEO(u) > PEO(w_k)$, we conclude $u \in N_h(w_k)$. □

Lemma 4. *Let $A \subset V$ such that $G(A)$ is connected. Let $x = highest(A)$ and $z = lowest(N(A))$. Then if $PEO(x) < PEO(z)$, $N(A) = N_h(x)$.*

Proof : First note that since $x = highest(A)$, $N_h(x) \bigcap A = \phi$ and therefore, $N_h(x) \subseteq N(A)$. Now we will prove $N(A) \subseteq N_h(x)$, from which we can conclude $N(A) = N_h(x)$. Let $y \in N(A)$. Then there is a $w \in A$ such that $y \in N(w)$. Also $PEO(y) \geq PEO(z) > PEO(x)$. Consider a chordless path $P = (w, \cdots, x)$, which is completely in $G(A)$. Such a path exists, since $G(A)$ is connected. Also since $x = highest(P)$, by corollary 1, P is an increasing path. Then by lemma 3, $y \in N_h(x)$. Thus we conclude that $N(A) \subseteq N_h(x)$. It follows that $N(A) = N_h(x)$. □

Lemma 5. *Let x be an interior vertex in a chordal graph G, ie $d_h(x) < n - PEO(x)$. Then $N_h(x)$ is a separator of G. In particular, $N_h(x)$ is a $(x - y)$ separator for all $y \notin N_h(x)$ with $PEO(y) > PEO(x)$.*

Proof: Let $PEO(x) = t$. Consider the induced subgraph $G(V_t)$ on the set of nodes $V_t = \{u \in V : PEO(u) \leq t\}$. Let C_i be the connected component of $G(V_t)$ which contains x. Obviously $x = highest(C_i)$. Since $N(C_i) \subseteq V - V_t$, clearly every node $z \in N(C_i)$ has $PEO(z) > t = PEO(x)$. Now by lemma 4, $N(C_i) = N_h(x)$. Since $d_h(x) < n - PEO(x)$, there should be a node y such that $PEO(y) > PEO(x)$ and $y \notin N_h(x)$. Clearly $y \notin C_i$ since $x = highest(C_i)$. Therefore $N_h(x)$ separates x from y, from which the lemma follows. □

Lemma 6. *Let $\kappa(G)$ be the connectivity of a chordal graph G. Then the number of interior nodes, $|V'|$ is at most $n - \kappa(G) - 1$.*

Proof: Let x be an interior node of G. Note that by lemma 5, $N_h(x)$ is a $(x - y)$ separator for some y with $PEO(y) > PEO(x)$. Then if $PEO(x) \geq n - \kappa(G)$, $|N_h(x)| < \kappa(G)$, which contradicts the definition of $\kappa(G)$. Thus we infer that $PEO(x) \leq n - \kappa(G) - 1$. It follows that $|V'| \leq n - \kappa(G) - 1$. □

Theorem 1. (Characterisation of minimum separators) *Let G be a chordal graph with $\kappa(G) = k$. Let M be the set of all minimum sized separators of G. Then*

$$M = \{S \subset V : S = N_h(x) \quad for\ some\ x \in V'\ such\ that\ d_h(x) = k\}$$

where $V' = \{u \in V : d_h(u) < n - PEO(u)\}$, the set of interior nodes of G.

Proof: First we prove that every $S \in M$ is indeed a minimum separator of G. By lemma 5, S is a separator of G. Since we have $k = d_h(x) = |N_h(x)| = |S|$, S is a minimum separator, since $\kappa(G) = k$.

Now let S be a minimum separator, ie $|S| = k$. Also let V be partitioned into A, S and B by S. Let $x = highest(A)$ and let $y = highest(B)$. Without loss of generality let $PEO(x) < PEO(y)$. Then x is an interior vertex of G, since $y \notin N(x)$ and $PEO(y) > PEO(x)$. By lemma 5, $N_h(x)$ is a separator. Also since $\kappa(G) = k$, $|N_h(x)| \geq k$. Clearly $N_h(x) \subseteq S$, since $x = highest(A)$. Then, since $|S| = k$, we conclude that $N_h(x) = S$, and $d_h(x) = k$. Thus $S \in M$. □.

Corollary 3. *Let G be a chordal graph, which is not a clique. Then $\kappa(G) = min_{u \in V'} d_h(u)$, where $V' = \{u \in V : d_h(u) < n - PEO(u)\}$, the set of interior nodes.*

Algorithm to enumerate minimum separators
Input : A connected chordal graph G, which is not complete.
Step 1: Compute a PEO of G.
Step 2: Find $d_h(u)$ for each $u \in V$.
Step 3: Let $V' = \{u : d_h(u) < n - PEO(u)\}$.
Step 4: $\kappa(G) = min_{u \in V'} d_h(u)$.

Step 5: For each $u \in V'$, if $d_h(u) = \kappa(G)$ output $N_h(u)$.

It can easily be verified that the above algorithm runs in $O(m + n)$ time. Computing the PEO takes $O(m+n)$ time [21]. Steps 2 and 5 can be implemented in $O(m)$ time, since it only involves traversing the neighbour lists of each node. Steps 3 and 4 takes $O(n)$ time.

The correctness of step 4 is ensured by corollary 3 and the correctness of step 5, namely the sets output are exactly the minimum separators is ensured by Theorem 1.

3 The Minimal Separators

Theorem 2. (Characterisation of minimal separators) *S is a minimal separator of a chordal graph G if and only if there exist two nonadjacent vertices $a, b \in V$, such that $PEO(a) < PEO(b)$, and $S = N_h(a) = N_h(a) \bigcap N(b)$.*

Proof: First assume that there are two nonadjacent nodes a, b with $PEO(a) < PEO(b)$, such that $S = N_h(a) = N_h(a) \bigcap N(b)$. Clearly a is an interior node and $S = N_h(a)$ is a separator which separates a from b, by lemma 5. Clearly S is minimal since $S = N_h(a) \bigcap N(b)$: removal of any proper subset of S leaves out a path from a to b.

Next, let S be a minimal separator which separates v_1 from v_2. Let A and B be the connected components of $G(V - S)$, which contains v_1 and v_2 respectively. First we show that there exists a node $a \in A$ and a node $b \in B$ such that $S = N(a) \bigcap N(b)$.

First note that $S = N(A) = N(B)$. To see this, observe that because of the minimality of S, $S \subseteq N(A)$. Clearly since A is a connected component of $G(V - S)$, $N(A) \subseteq S$, also. Thus we have $S = N(A)$. By similar arguments $S = N(B)$.

Let $z = lowest(S)$, and $x = highest(A)$. If $PEO(x) < PEO(z)$, then since $N(A) = S$, by lemma 4, we have $S = N(A) = N_h(x) \subseteq N(x)$. In this case let $a = x$.

If $PEO(x) > PEO(z)$, consider a chordless path from z to x, which is completely in the induced subgraph $G(A \bigcup \{z\})$. (Observe that such a path exists, because $G(A \bigcup \{z\})$ is connected since $z \in N(A)$). Let u be the second node in this chordless path starting from z. By corollary 2, $PEO(u) > PEO(z)$. Also by lemma 1, S forms a complete subgraph of G. Thus we have $S - \{z\} \bigcup \{u\} \subseteq N_h(z)$, since $z = lowest(S)$. By definition of PEO, $N_h(z) \bigcup \{z\}$ form a complete subgraph and thus we have $S \subseteq N(u)$. In this case let $a = u$.

By similar arguments as above, we get a node $b \in B$ such that $S \subseteq N(b)$. Now we show that $S = N(a) \bigcap N(b)$. Clearly $S \subseteq N(a) \bigcap N(b)$. Suppose there is a node $v \in (N(a) \bigcap N(b) - S)$. Then S can not separate a from b, which is a contradiction. Thus we have $S = N(a) \bigcap N(b)$.

Without loss of generality, assume that $PEO(a) < PEO(b)$. We will now show that $S = N_h(a) \bigcap N(b)$. First we claim that $PEO(a) < PEO(z)$. Otherwise let $PEO(b) > PEO(a) > PEO(z)$. Note that since a and b are neighbours of every node in S, we have $a, b \in N_h(z)$. By definition of PEO, there will be an

edge between a and b which is a contradiction since S separates a from b. Thus $PEO(a) < PEO(z)$, and we have $S \subseteq N_h(a)$. It follows that $S = N_h(a) \bigcap N(b)$.

Finally, since we have established $PEO(a) < PEO(z)$, it follows that $a = x = highest(A)$. (Remember we chose a, from two choices: one was x, the $highest(A)$ and the other was u, the second node in the chordless path from z to x. In the latter case, $PEO(a) > PEO(z)$). Thus we have $S = N_h(a)$.

□

The above theorem says that every minimal separator has to be $N_h(u)$ for some interior vertex. Our algorithm to enumerate all minimal separators examines $N_h(u)$ for each interior vertex u, and decides whether it is a minimal separator or not. We consider a node w to be a *witness* for $N_h(u)$, if the existence of w proves that $N_h(u)$ is a minimal separator. For example, if we can find a node w, nonadjacent to u with $PEO(w) > PEO(u)$ and $N(w) \supseteq N_h(u)$, then by Theorem 2, w is a witness for $N_h(u)$. Thus the issue in designing the algorithm is to efficiently identify a witness, if one exists.

In our algorithm, we make use of only two types of witnesses. These types are defined in terms of a special node in $N_h(u)$, namely $z = lowest(N_h(u))$.

Definition 6. *First type witness : A node w is defined to be a first type witness for $N_h(u)$, iff $w \in N_h(z) - N(u)$.*

Lemma 7. (First type witnesses are indeed witnesses) *If there exists a node $w \in N_h(z) - N(u)$, then $N_h(u)$ is a minimal separator.*

Proof: First note that $w \in N_h(z) - N(u)$ is not adjacent to u and since $w \in N_h(z)$, $PEO(w) > PEO(u)$. Now note that since $N_h(u)$ is a clique, and $z = lowest(N_h(u))$, we have $N_h(u) - \{z\} \subseteq N_h(z)$. Also $w \in N_h(z)$. Since by definition of PEO, $\{z\} \bigcup N_h(z)$ forms a clique, it follows that $N_h(u) \subseteq N(w)$. Then by Theorem 2, w is a witness for $N_h(u)$. □

Definition 7. *Second type witness: A node $w \neq u$, is defined to be a second type witness iff $w \in N_l(z) - N(u)$, and $N_h(w) = N_h(u)$.*

Lemma 8. (Second type witnesses are indeed witnesses) *If there exists $w \neq u$, $w \in N_l(z) - N(u)$, such that $N_h(w) = N_h(u)$, then $N_h(u)$ is a minimal separator.*

Proof: First note that w is nonadjacent to u. If $PEO(w) > PEO(u)$, clearly since $N_h(u) \bigcap N(w) = N_h(w) = N_h(u)$, by Theorem 2, $N_h(u)$ is a minimal separator and w is a witness for $N_h(u)$. If $PEO(w) < PEO(u)$, we just have to interchange the roles of u and w, and we again get $N_h(u) = N_h(w)$ is a minimal separator. □

The first and second type witnesses certainly do not exhaust the set of all possible witnesses. But it turns out that these two types are sufficient for our algorithm. This is because of the following lemma, which assures that, if first type witnesses are not available, then a second type witness is guaranteed to exist, provided the $N_h(u)$ in question is a minimal separator.

Lemma 9. *Let $N_h(u)$ be a minimal separator. If $N_h(z) - N(u) = \phi$, then there exists a node $w \neq u$, such that $w \in N_l(z) - N(u)$, and $N_h(w) = N_h(u)$.*

Proof: Since $N_h(u)$ is a minimal separator, there exists a node $v \in V$, such that $N_h(u)$ minimally separates u from v. (If $N_h(u)$ is a minimal $(v_1 - v_2)$ separator, at least one of them, either v_1 or v_2, should be in a different connected component than that of u, when $N_h(u)$ is removed from G. Then clearly $N_h(u)$ minimally separates u from that node). Let A be the connected component of $G(V - N_h(u))$ which contains v. Clearly $u \notin A$. First note that by minimality of the separator $N_h(u)$, $N_h(u) = N(A)$. Let $x = highest(A)$. We claim that $PEO(x) < PEO(z)$. Otherwise, if $PEO(x) > PEO(z)$, consider a chordless path $P = (z, w_1, \cdots, x)$ in $G(A \bigcup \{z\})$, which is guaranteed to exist since $G(A \bigcup \{z\})$ is connected. Then clearly $w_1 \in N_h(z)$, by corollary 2. Also $w_1 \notin N(u)$, since $w_1 \in A$ and u is separated from A by $N_h(u)$. Thus $w_1 \in N_h(z) - N(u)$, contradicting the assumption that $N_h(z) - N(u) = \phi$. We infer that $PEO(x) < PEO(z)$. We conclude from lemma 4, that $N_h(x) = N(A) = N_h(u)$. Also since $x \in N_l(z)$ and $x \notin N(u)$, we have $x \in N_l(z) - N(u)$. The lemma follows. □

Finally, the reason why we consider these special kind of witnesses, namely the second type witnesses, is exactly that, they are easy to identify. The following lemma explains this.

Lemma 10. *Suppose that $N_h(z) - N(u) = \phi$. A node $w \neq u$, $w \in N_l(z) - N(u)$ is a second type witness if and only if $d_h(w) = d_h(u)$ and $lowest(N_h(w)) = z$.*

Proof : If w is a second type witness, ie, if $N_h(w) = N_h(u)$, clearly we have $d_h(u) = d_h(w)$ and $lowest(N_h(w)) = z$. Now suppose that there is a node $w \in N_l(z) - N(u)$, with $d_h(u) = d_h(w)$ and $lowest(N_h(w)) = z$. We first show that $N_h(w) \subseteq N_h(u)$. Let $y \in N_h(w)$. Noting that by the assumption $z = lowest(N_h(w))$, $z \in N_h(w)$ also, we get $(z, y) \in E$, since $N_h(w)$ should form a clique. But $z = lowest(N_h(w))$, therefore $PEO(y) > PEO(z)$, ie $y \in N_h(z)$. Now if $y \notin N_h(u)$, then $y \in N_h(z) - N_h(u) = N_h(z) - N(u)$, which contradicts the assumption that $N_h(z) - N(u) = \phi$. Thus $y \in N_h(u)$. It follows that $N_h(w) \subseteq N_h(u)$. But since $d_h(u) = d_h(w)$, it has to be the case that $N_h(u) = N_h(w)$. That is, w is a second type witness of $N_h(u)$. □

Algorithm To Enumerate Minimal Separators

Step 1: Find a PEO of G.

Step 2: (Preprocessing:) For each node $u \in V$, find $d_h(u)$ and $z_u = lowest(N_h(u))$. Store these information with the node, for future use.

Step 3: For each $u \in V'$ with nonempty $N_h(u)$, do:

Step 3.1: (Looking for first type witness :) If $N_h(z_u) - N(u) \neq \phi$, output $N_h(u)$.

Step 3.2: (Looking for second type witness :) If $N_h(z_u) - N(u) = \phi$, then if there is a node $w \neq u$ and $w \in N_l(z_u) - N(u)$, such that $d_h(u) = d_h(w)$ and $z_w = z_u$, then output $N_h(u)$.

Theorem 3. *The above algorithm outputs exactly the set of all minimal separators.*

Proof: First note that the algorithm outputs $N_h(u)$ only if a witness is found. Therefore it outputs only minimal separators. Now every minimal separator is $N_h(u)$ for some $u \in V'$, by Theorem 2, and has either a first type witness or second type witness by lemma 9. Therefore every minimal separator will be output. □

Theorem 4. *The above algorithm runs in $O(m+n)$ time.*

Proof: Note that implementing step 2 and 3 only involves traversing the list of neighbours of each node a constant number of times. Since $\Sigma_{u \in V} d(u) = 2m$, this can be done in $O(m)$ time. Step 1 can be implemented in $O(m+n)$ time. □

4 Upper Bound for Number of Minimal Separators of Chordal Graphs

Theorem 5. *Number of minimum separators ≤ Number of minimal separators ≤ $n - \kappa(G) - 1$.*

Proof : Since every minimum separator is a minimal separator, we just have to show the second inequality. It is clear from Theorem 2 that each minimal separator is $N_h(u)$ for some interior vertex u. But the number of interior vertices is $\leq n - \kappa(G) - 1$, by lemma 6. Hence the result. □

Finally we show that the above inequality can be tight, by constructing a graph $G_n(V_n, E_n)$, with $\kappa(G) = k$, and $n \geq k+2$, for $k \geq 1$, such that the number of minimum separators $= n-k-1$. This clearly proves that for G_n, the number of minimum separators = number of minimal separators $= n - \kappa(G) - 1$. For this we just have to show that V'_n, the set of interior nodes of G_n, contain exactly $n-k-1$ nodes, and for each $u \in V'_n$, $d_h(u) = k$ (see Theorem 1).

First let $l = k+2$. Let $V_l = \{v_1, v_2, \cdots, v_l\}$. Define $G_l(V_l, E_l)$ to be the graph obtained by removing the edge (v_1, v_l) from the complete graph on V_l. Consider an ordering f_l of G_l defined by $f_l(v_i) = i$, for $1 \leq i \leq l$. We claim that G_l is a chordal graph, and f_l is a PEO. We leave the details to the reader. Also it is easy to verify that $\kappa(G) = k$ and $V'_l = \{v_1\}$. Thus we have just one, minimum separator for G_l.

Now we inductively construct $G_n(V_n, E_n)$. Let $G_{n-1}(V_{n-1}, E_{n-1})$ be a chordal graph with PEO defined by $f_{n-1}(v_i) = i$, for $1 \leq i \leq n-1$. Let $\kappa(G_{n-1}) = k$ and let $V'_{n-1} = \{v_1, v_2, \cdots, v_{n-k-2}\}$. Also let for all $u \in V'_{n-1}$, $d_h(u) = k$ and $N_h(u)$ distinct from one another. In other words, the number of minimum separators of G_{n-1} is exactly $(n-1) - k - 1$.

Now let $V_n = V_{n-1} \bigcup \{v_n\}$, where v_n is a new node. Let $f_n(v_i) = i$, for $1 \leq i \leq n$ and $E_n = E_{n-1} \bigcup \{(v_i, v_n) : v_i \in V_{n-1}$ *with* $f_n(v_i) = i > n-k-1\}$. We claim that G_n is a chordal graph with a PEO defined by f_n. Also it is not difficult to show that $\kappa(G_n) = k$. We leave the details to the reader.

Clearly $V'_n = V'_{n-1} \bigcup \{v_{n-k-1}\}$. Note that $d_h(v_{n-k-1}) = k$. Thus we have $|V'_n| = n-k-1$, and $\forall v \in V'_n, d_h(v) = \kappa(G)$ and $N_h(v_{n-k-1}) \neq N_h(u) \forall u \in V'_{n-1}$. Thus G_n has exactly $n - \kappa(G) - 1$ minimum separators by Theorem 1.

References

1. H. Ariyoshi, *Cut-set graph and systematic generation of separating sets.* IEEE Transactions on circuit theory, CT-19 (1972) 233-240.
2. G.A.Dirac, *On rigid circuit graphs.* Abh. Math. Sem. Univ. Hamburg, 25:71-76,1961.
3. S.Even, R.E.Tarjan, *Network Flow And Testing Graph Connectivity* SIAM J. Comp. 4(4) 507-518, December 1975.
4. H.N.Gabow, *Using Expander Graphs to Find Vertex Connectivity.* Annual Symposium on Foundations of Computer Science, 2000.
5. L.A.Goldberg, *Efficient Algorithms For Listing Combinatorial Structures.* Cambridge University Press, Cambridge, 1993.
6. M.C.Golumbic, *Algorithmic Graph Theory And Perfect Graphs.* Academic Press, New York, 1980.
7. F.Harary, Graph Theory. Addison-Wesley Reading, MA, 1969.
8. Hong Shen and Weifa Liang, *Efficient Enumeration Of All Minimal Separators In A Graph* Theoretical Computer Science, 180(1-2) 169-180, 10 June 1997
9. A.Kanevsky, *Finding All Minimum Size Separating Vertex Sets In Graphs.* NETWORKS, 23 533-541,1993.
10. T.Kloks and D.Kratsch, *Listing All Minimal Separators Of A Graph* SIAM J. Comput. 27(3) 605-613 June 1998
11. T.Kloks, H.Bodlaender and D.Kratsch, *Treewidth And Path Width Of Permutation Graphs* SIAM J. Descrete Math. 8 , 606-616, 1995.
12. T.Kloks, *Treewidth - Computations And Approximations.* Lecture Notes In Computer Science 842, Springer Verlag, Berlin 1994.
13. T.Kloks, *Treewidth Of Circle Graphs.* Int. J. Of Foundations In Comp. Sci., 7 111-120 , 1996.
14. T.Kloks and D.Kratsch, *Treewidth Of Chordal Bipartite Graphsa.* J. Algorithms, 19, 266-281 1995.
15. R.H.Mohring, *Graph Problems Related To Gate Matrix Layout And PLA Folding.* Computational Graph Theory, Pages 17-52, Springer, Wein, New York,1990.
16. Monika R Henzinger, *A Static 2-Approximation Algorithm For Vertex Connectivity And An Incremental Approximation Algorithm For Edge And Vertex Connectivity.* J. of Algorithms 24(1) 194-220 July 1997
17. Monika R Henzinger,Satish Rao, and H.N.Gabow, *Computing Vertex Connectivity: New Bounds From Old Techniques.* 37th Anual Symposium On Foundations Of Computer Science, 462-471 Oct. 1996.
18. Samir Khuller and Baruch Schieber, *k-Connectivity And Finding Disjoint s-t Paths In Graphs.* SIAM J. Comp. 20(2) 352-375 April 1991
19. Shimon Even, *An Algorithm For Determining Whether The Connectivity Of A Graph Is At Least k* . SIAM J. Comp. 4(3) 393-396 September 1975.
20. M.Yannakakis, *Computing The Minimum Fill-in Is NP-Complete* SIAM J. on Alge. Discre. Math. 2 77-79, 1981.
21. M.Yannakakis and R.E.Tarjan, *Simple Linear Time Algorithms To Test Chordality of Graphs, Test Acyclicity of hypergraphs and Selectively Reduce Acyclic Hypergraphs.* SIAM J. Comput.13,566-579, 1984
22. Zvi Galil, *Finding The Vertex Connectivity Of Graphs.* SIAM J. Compu. 9(1) 197-199 Feb 1980.

Graph Separators: A Parameterized View*

Jochen Alber**, Henning Fernau, and Rolf Niedermeier

Universität Tübingen, Wilhelm-Schickard-Institut für Informatik,
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany,
{alber,fernau,niedermr}@informatik.uni-tuebingen.de

Abstract. Graph separation is a well-known tool to make (hard) graph problems accessible for a divide and conquer approach. We show how to use graph separator theorems in order to develop fixed parameter algorithms for many well-known NP-hard (planar) graph problems. We coin the key notion of glueable select&verify graph problems and derive from that a prospective way to easily check whether a planar graph problem will allow for a fixed parameter divide and conquer algorithm of running time $c^{\sqrt{k}} \cdot n^{O(1)}$ for a constant c.

1 Introduction

Algorithm designers are often faced with problems which, when viewed from classical computational complexity theory, are "intractable." More formally speaking, these problems can be shown to be *NP*-hard. In many applications, however, a certain part (called the *parameter*) of the whole problem can be identified which tends to be of small size k when compared with the size n of the whole problem instance. This leads to the study of parameterized complexity [7].

Fixed parameter tractability. Formally, one terms a (parameterized) problem *fixed parameter tractable* if it allows for a solving algorithm running in time $f(k)n^{O(1)}$ on input instance (I, k), where $n = |I|$ and f is an arbitrary function only depending on k. We will also term such algorithms "$f(k)$-algorithms" for brevity, focusing on the exponential part of the running time bound. The associated complexity class is called FPT. Of course, designing fixed parameter algorithms with a "small" function f is desirable. To our knowledge, so far, only one non-trivial fixed parameter tractability result where the corresponding function f is sublinear in the exponent, namely $f(k) = c^{\sqrt{k}}$ [1], is known: PLANAR DOMINATING SET. Similar results hold for closely related problems such as FACE COVER, PLANAR INDEPENDENT DOMINATING SET, PLANAR WEIGHTED DOMINATING SET, etc. [1]. In the companion paper [2], we proved similar results for a much broader class of planar graph problems, presenting a general methodology based on concepts such as tree decompositions and bounded outerplanarity.

* Full version: Technical Report No. WSI–2001–8, Universität Tübingen (Fed. Rep. of Germany), Wilhelm-Schickard-Institut für Informatik.

** Supported by the Deutsche Forschungsgemeinschaft (research project PEAL (Parameterized complexity and Exact ALgorithms), NI 369/1-1).

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 318–327, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Here, we will also discuss a rather general approach for obtaining parameterized graph algorithms running in time[1] $O(c^{e(k)} \cdot q(n))$ for sublinear functions e, i.e., $e(k) \in o(k)$. More precisely, we investigate the use of (planar) separator theorems in this context, yielding an alternative and conceptually rather different framework in comparison with [2].

General outline. It is a crucial goal throughout the paper not to narrowly stick to problem-specific approaches, but to try to widen the techniques as far as possible. More specifically, we show how to use separator theorems for different graph classes, such as, e.g., the well-known planar separator theorem due to Lipton and Tarjan [8], in combination with known algorithms for obtaining (linear size) problem kernels in order to obtain fixed parameter divide and conquer algorithms. Special care is taken for the dependency of the running time on the "graph separator parameters" (and how they influence the recurrences in the running time analysis). We consider a broad class of problems that can be attacked by this approach, namely, in principle, all so-called glueable select&verify problems such as, e.g., PLANAR DOMINATING SET.[2] Also, we exhibit the influence on the running time analysis of so-called cycle separators. Although the constants achieved in our setting so far seem to be too large in order to yield practical algorithms, our approach provides a general, sound, mathematical formalization of a rich class of problems that allow for divide and conquer fixed parameter algorithms. Our methodology seems to leave much room for improvement in many directions. For instance, we introduce the novel concept of problem cores that can replace problem kernels in our setting. Finally, we give a push to the study of subclasses of the parameterized complexity class FPT. In this sense, our work also might serve as a starting point for more algorithmic (also concerning graph theory with respect to separator theorems), as well as more structural complexity-theoretic lines of future research in parameterized complexity.

Due to the lack of space, several details are deferred to the long version.

2 Basic Definitions and Preliminaries

We consider undirected graphs $G = (V, E)$, V denoting the vertex set and E denoting the edge set. We only consider simple graphs (i.e., with no double edges) without self-loops. Sometimes, we refer to V by $V(G)$ in order to emphasize that V is the vertex set of graph G; by $N(v)$ we refer to the set of vertices adjacent to v. $G[D]$ denotes the subgraph induced by a vertex set D. For graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, by $G_1 \cap G_2$, we denote the graph $(V_1 \cap V_2, E_1 \cap E_2)$. $G' = (V', E')$ is a subgraph of $G = (V, E)$, denoted by $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. In this paper, we only consider graph classes, denoted by $\mathbb{G}$, that are closed under taking subgraphs. The most important among these graph classes is that of *planar* graphs, i.e., graphs that have a drawing in the plane without

[1] Here and in the following, n is the number of vertices of the input graph and k is the parameter of the considered graph problem.

[2] Lipton and Tarjan [9] only describe a solution for the structurally much simpler PLANAR INDEPENDENT SET in detail.

edge crossings. Among others, we study the following vertex subsets of a graph $G = (V, E)$: A *vertex cover* $C \subseteq V$ satisfies that every edge of G has at least one endpoint in C. An *independent set* is a set of pairwise nonadjacent vertices. A *dominating set* $D \subseteq V$ obeys that each of the rest of the vertices in G has at least one neighbor in D. The corresponding problems are denoted by (PLANAR) VERTEX COVER, INDEPENDENT SET, and DOMINATING SET.
Further conventions: we write $A+B$ to denote the disjoint union of sets A and B. We let $0 \cdot (\pm\infty) = 0$.

Linear problem kernels. Let $\mathcal{L}$ be a parameterized problem, i.e., $\mathcal{L}$ is a subset of $\Sigma^* \times \mathbb{N}$.[3] *Reduction to problem kernel*, then, means to replace instance $(I, k) \in \Sigma^* \times \mathbb{N}$ by a "reduced" instance $(I', k') \in \Sigma^* \times \mathbb{N}$ (which we call *problem kernel*) such that $k' \leq c \cdot k, |I'| \leq p(k)$ with constant c, some function p only depending on k, and $(I, k) \in \mathcal{L}$ iff $(I', k') \in \mathcal{L}$. Furthermore, we require that the reduction from (I, k) to (I', k') is computable in polynomial time $T_K(|I|, k)$.

Often, the best one can hope for is that the problem kernel is size linear in k, a so-called *linear problem kernel*. For instance, using a theorem of Nemhauser and Trotter, Chen *et al.* [4] observed a problem kernel of size $2k$ for VERTEX COVER on general (not necessarily planar) graphs. Furthermore, PLANAR INDEPENDENT SET has a problem kernel of size $4k$ due to the four color theorem. Once having a linear size problem kernel, it is fairly easy to use our framework to get $c^{\sqrt{k}}$-algorithms for these problems based upon the famous planar separator theorem [8,9]. The constant factor in the problem kernel size directly influences the value of the exponential base. Hence, lowering the kernel size is a crucial goal.

Classical separator theorems. Let $G = (V, E)$ be an undirected graph. A *separator* $S \subseteq V$ of G partitions V into two *parts* $A_1 \subseteq V$ and $A_2 \subseteq V$ such that $A_1 + S + A_2 = V$, and no edge joins vertices in A_1 and A_2; (A_1, S, A_2) is called a *separation* of G. When we restrict our attention to *planar graphs*, S is called *cycle separator* if it forms a cycle in some triangulation of G. According to Lipton and Tarjan [8], an *$f(\cdot)$-separator theorem (with constants $\alpha < 1$, $\beta > 0$)* for a class $\mathbb{G}$ of graphs which is closed under taking subgraphs is a theorem of the following form: If G is any n-vertex graph in $\mathbb{G}$, then there is a separation (A_1, S, A_2) of G such that neither A_1 nor A_2 contains more than αn vertices, and S contains no more than $\beta f(n)$ vertices.

Stated in this framework, the planar separator theorem due to Lipton and Tarjan [8] is a $\sqrt{\cdot}$-separator theorem with constants $\alpha = 2/3$ and $\beta = 2\sqrt{2}$. The current record for $\alpha = 2/3$ is $\beta \approx 1.97$ [6]. Similar $\sqrt{\cdot}$-separator theorems are also known for other graph classes, e.g., for the class of graphs of bounded genus, see [5]. It is also possible to incorporate *weights* in most separator theorems. We refer to them as $f(\cdot)$-separator theorems for weighted graphs.

[3] In this paper, we assume the parameter to be a positive integer although, in general, it might also be from an arbitrary language (e.g., being a subgraph).

3 Glueable Graph Problems

Select&verify graph problems. A set $\mathcal{G}$ of tuples (G, k), G an undirected graph with vertex set $V = \{v_1, \ldots, v_n\}$ and k a nonnegative real number, is called a *select&verify (graph) problem* if there exists a pair $(P_\cdot, \text{opt})$ with $\text{opt} \in \{\min, \max\}$, such that $P_\cdot$ is a function that assigns to G a polynomial time computable function of the form $P_G = P_G^{\text{sel}} + P_G^{\text{ver}}$, where $P_G^{\text{sel}} : \{0,1\}^n \to \mathbb{R}_+$, $P_G^{\text{ver}} : \{0,1\}^n \to \{0, \pm\infty\}$, and

$$(G,k) \in \mathcal{G} \quad \Leftrightarrow \quad \begin{cases} \text{opt}_{\boldsymbol{x}\in\{0,1\}^n} P_G(\boldsymbol{x}) \leq k & \text{if opt} = \min, \\ \text{opt}_{\boldsymbol{x}\in\{0,1\}^n} P_G(\boldsymbol{x}) \geq k & \text{if opt} = \max. \end{cases}$$

For $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ with $P_G(\boldsymbol{x}) \leq k$ if opt $= \min$ and with $P_G(\boldsymbol{x}) \geq k$ if opt $= \max$, the vertex set *selected* by $\boldsymbol{x}$ and *verified* by P_G is $\{v_i \in V \mid x_i = 1, 1 \leq i \leq n\}$. A vector $\boldsymbol{x}$ is called *admissible* if $P_G^{\text{ver}}(\boldsymbol{x}) = 0$.

The intuition behind the term $P_\cdot = P_\cdot^{\text{sel}} + P_\cdot^{\text{ver}}$ is that the "selecting function" $P_\cdot^{\text{sel}}$ counts the size of the selected set of vertices and the "verifying function" $P_\cdot^{\text{ver}}$ verifies whether this choice of vertices is an admissible solution. Every select&verify graph problem that additionally admits a problem kernel of size $p(k)$ is solvable in time $O(2^{p(k)}p(k) + T_K(n,k))$.

We now give two examples for select&verify problems by specifying the function $P_G = P_G^{\text{sel}} + P_G^{\text{ver}}$. In both cases the "selecting function" for a graph $G = (V, E)$ will be $P_G^{\text{sel}} = \sum_{v_i \in V} x_i$. Firstly, in the case of VERTEX COVER, we have opt $=$ min and choose $P_G^{\text{ver}}(\boldsymbol{x}) = \sum_{\{v_i,v_j\}\in E} \infty \cdot (1-x_i)(1-x_j)$. Thus, $P_G(\boldsymbol{x}) \leq k$ guarantees a size at most k vertex cover set. Secondly, for DOMINATING SET, we have $P_G^{\text{ver}}(\boldsymbol{x}) = \sum_{v_i \in V}(\infty \cdot (1-x_i) \cdot \prod_{\{v_i,v_j\}\in E}(1-x_j))$.

We will also need a notion of select&verify problems where the "selecting function" and the "verifying function" operate on a subgraph of the given graph: Let $P_\cdot = P_\cdot^{\text{sel}} + P_\cdot^{\text{ver}}$ be the function of a select&verify problem. For an n-vertex graph G and subgraphs $G^{\text{ver}} = (V^{\text{ver}}, E^{\text{ver}})$, $G^{\text{sel}} = (V^{\text{sel}}, E^{\text{sel}}) \subseteq G$, we let

$$P_{G^{\text{ver}}}(\boldsymbol{x} \mid G^{\text{sel}}) := P_{G^{\text{ver}}}^{\text{ver}}(\pi_{V^{\text{ver}}}(\boldsymbol{x})) + P_{G^{\text{sel}}}^{\text{sel}}(\pi_{V^{\text{sel}}}(\boldsymbol{x})),$$

where $\pi_{V'}$ is the projection of the vector $\boldsymbol{x} \in \{0,1\}^n$ to the variables corresponding to the vertices in $V' \subseteq V$.

Glueability. We are going to solve graph problems, slicing the given graph into small pieces with the help of small separators. The separators will serve as boundaries between the different graph parts into which the graph is split. For each possible assignment of the vertices in the separators, we want to – independently – solve the corresponding problems on the graph parts and then reconstruct a solution for the whole graph by "gluing" together the solutions for the graph parts. We need to assign *colors* to the separator vertices in the course of the algorithm. Hence, our algorithm has to be designed in such a manner that it can also cope with colored graphs. In general (e.g., in the case of the DOMINATING SET problem), it is not sufficient to simply use the two colors 1 (for encoding "in the selected set") and 0 (for "not in the selected set").

Let us introduce some auxiliary notions. Let $G = (V, E)$ be an undirected graph and let C_0, C_1 be finite, disjoint sets. A C_0-C_1-*coloring* of G is a function $\chi : V \to C_0 + C_1 + \{\#\}$.[4] For $V' \subseteq V$, a function $\chi : V' \to C_0 + C_1$ can naturally be extended to a C_0-C_1-coloring of G by setting $\chi(v) = \#$ for all $v \in V \setminus V'$.

Consider a vector $\boldsymbol{x} \in \{0,1\}^{|V|}$. Let χ be a C_0-C_1-coloring of G. Then, $\boldsymbol{x}$ is *consistent* with χ, written $\boldsymbol{x} \sim \chi$, if, for $i = 0, 1$ and $j = 1, \ldots, |V|$, $\chi(v_j) \in C_i \;\Rightarrow\; x_j = i, .$

If χ is a C_0-C_1-coloring of G and if χ' is a C_0'-C_1'-coloring of G, then χ is *preserved* by χ', written $\chi \rightsquigarrow \chi'$, if $\forall v \in V\, \forall i \in \{0,1\}\, (\chi(v) \in C_i \Rightarrow \chi'(v) \in C_i')$.

In the next section, when doing the divide and conquer approach with a given separator, we will deal with colorings on two different color sets: one color set $C^{\,\mathrm{int}} := C_0^{\,\mathrm{int}} + C_1^{\,\mathrm{int}} + \{\#\}$ of *internal* colors that will be used for the assignments of colors to the separator vertices and a color set $C^{\mathrm{ext}} := C_0^{\mathrm{ext}} + C_1^{\mathrm{ext}} + \{\#\}$ of *external* colors that will be used for handing down the information in the divide-step of the algorithm. The idea is that, in each recursive step, we will be confronted with a graph "pre-colored" with external colors. Every function $\oplus$ that assigns to a pair $(\chi^{\mathrm{ext}}, \chi^{\mathrm{int}})$ with $\chi^{\mathrm{ext}} : V \to C^{\mathrm{ext}}$, $\chi^{\mathrm{int}} : V \to C^{\,\mathrm{int}}$, $\chi^{\mathrm{ext}} \rightsquigarrow \chi^{\mathrm{int}}$, a ($C_0^{\mathrm{ext}}$-$C_1^{\mathrm{ext}}$)-coloring $\chi^{\mathrm{ext}} \oplus \chi^{\mathrm{int}}$ is called a *recoloring* if $\chi^{\mathrm{int}} \rightsquigarrow \chi^{\mathrm{ext}} \oplus \chi^{\mathrm{int}}$.

From the point of view of recursion, χ^{ext} is the pre-coloring which a certain recursion instance "receives" from the calling instance and χ^{int} represents a coloring which this instance assigns to a certain part of the graph. The coloring $\chi^{\mathrm{ext}} \oplus \chi^{\mathrm{int}}$ is handed down in the recursion.

We now introduce the central notion of "glueable" select&verify problems. This formalizes those problems that can be solved with separator based divide and conquer techniques as described above.

Definition 1. *A select&verify problem $\mathcal{G}$ given by $(P., \mathrm{opt})$ is* glueable with σ colors *if there exist*

- *a color set $C^{\,int} := C_0^{\,int} + C_1^{\,int} + \{\#\}$ of internal colors with $|C_0^{\,int} + C_1^{\,int}| = \sigma$;*
- *a color set $C^{ext} := C_0^{ext} + C_1^{ext} + \{\#\}$ of external colors;*
- *a polynomial time computable function $h : (\mathbb{R}_+ \cup \{\pm\infty\})^3 \to \mathbb{R}_+ \cup \{\pm\infty\}$;*

and, for every n-vertex graph $G = (V, E)$ and subgraphs $G^{ver}, G^{sel} \subseteq G$ with a separation (A_1, S, A_2) of G^{ver}, we find

- *recolorings $\oplus_X$ for each $X \in \{A_1, S, A_2\}$, and*
- *for each internal coloring $\chi^{int} : S \to C^{\,int}$, subgraphs $G^{ver}_{A_i}(\chi^{int})$ of G^{ver} with $G^{ver}[A_i] \subseteq G^{ver}_{A_i}(\chi^{int}) \subseteq G^{ver}[A_i + S]$ for $i = 1, 2$, and subgraphs $G^{ver}_S(\chi^{int})$ of G^{ver} with $G^{ver}_S(\chi^{int}) \subseteq G^{ver}[S]$*

such that, for each external coloring $\chi^{ext} : V \to C^{ext}$,

$$\begin{aligned} &\mathrm{opt}\{P_{G^{ver}}(\boldsymbol{x} \mid G^{sel}) \mid \boldsymbol{x} \in \{0,1\}^n \wedge \boldsymbol{x} \sim \chi^{ext}\} \\ &= \mathrm{opt}_{\substack{\chi^{int}: S \to C_0^{\,int} + C_1^{\,int} \\ \chi^{ext} \rightsquigarrow \chi^{int}}} h\big(Eval_{A_1}(\chi^{int}), Eval_S(\chi^{int}), Eval_{A_2}(\chi^{int})\big). \end{aligned} \tag{1}$$

[4] The symbol # will be used for the undefined (i.e., not yet defined) color.

Here, $Eval_X(\cdot)$ *for* $X \in \{A_1, S, A_2\}$ *is of the form* $Eval_X(\chi^{int}) =$

$$\mathrm{opt}\{P_{G_X^{ver}(\chi^{int})}(\boldsymbol{x} \mid G^{ver}[X] \cap G^{sel}) \mid \boldsymbol{x} \in \{0,1\}^n \wedge \boldsymbol{x} \sim (\chi^{ext} \oplus_X \chi^{int})\}.$$

Lemma 1. VERTEX COVER *and* INDEPENDENT SET *are glueable with 2 colors and* DOMINATING SET *is glueable with 4 colors.*

Proof. For VERTEX COVER, we use the color sets $C_i^\ell := \{i^\ell\}$ for $\ell \in \{\mathrm{int}, \mathrm{ext}\}$ and $i = 0, 1$. The function h is $h(x, y, z) = x+y+z$. The subgraphs $G_X^{\mathrm{ver}}(\chi^{\mathrm{int}})$ for $X \in \{A_1, S, A_2\}$ and $\chi^{\mathrm{int}} : S \to C_0^{\mathrm{int}} + C_1^{\mathrm{int}}$ are $G_X^{\mathrm{ver}}(\chi^{\mathrm{int}}) := G^{\mathrm{ver}}[X]$. In this way, the subroutine $\mathrm{Eval}_S(\chi^{\mathrm{int}})$ checks whether the coloring χ^{int} yields a vertex cover on $G^{\mathrm{ver}}[S]$ and the subroutines $\mathrm{Eval}_{A_i}(\chi^{\mathrm{int}})$ compute the minimum size vertex cover on $G^{\mathrm{ver}}[A_i]$. However, we still need to make sure that all edges going from A_i to S are covered. If a vertex in S is assigned a 1^{int} by χ^{int}, the incident edges are already covered. In the case of a 0^{int}-assignment for a vertex $v \in S$, we can color all neighbors in $N(v) \cap A_i$ to belong to the vertex cover. This is done by the following recolorings "$\oplus_{A_i}$." Define

$$(\chi^{\mathrm{ext}} \oplus_{A_i} \chi^{\mathrm{int}})(v) = \begin{cases} 0^{\mathrm{ext}} \text{ if } \chi^{\mathrm{int}}(v) = 0^{\mathrm{int}}, \\ 1^{\mathrm{ext}} \text{ if } \chi^{\mathrm{int}}(v) = 1^{\mathrm{int}} \text{ or } \exists w \in N(v) \text{ with } \chi^{\mathrm{int}}(w) = 0^{\mathrm{int}}, \\ \#, \quad \text{otherwise.} \end{cases}$$

By this recoloring definition, an edge between a separator vertex and a vertex in A_i which is not covered by the separator vertex (due to the currently considered internal coloring) will be covered by the vertex in A_i. Our above reasoning shows that – with these settings – equation (1) is satisfied.

INDEPENDENT SET is shown to be glueable with 2 colors by a similar idea.

Regarding DOMINATING SET, we refer to the full paper. □

We want to mention in passing that – besides the problems given in Lemma 1 – many more select&verify problems are glueable. In particular, this is true for the weighted versions and variations of the above mentioned problems.

4 Fixed Parameter Divide and Conquer Algorithms

For the considerations in this sections, let us fix a graph class $\mathbb{G}$ for which a $\sqrt{\cdot}$-separator theorem with constants α and β is known. Then, we consider a select&verify graph problem $\mathcal{G}$ defined by $(P_\cdot, \mathrm{opt})$ that is glueable with σ colors.

4.1 Using Glueability for Divide and Conquer

The evaluation of the term $\mathrm{opt}_{\boldsymbol{x} \in \{0,1\}^n} P_G(\boldsymbol{x})$ can be done recursively as follows. Start the computation with $\mathrm{opt}\{P_G(\boldsymbol{x}) \mid \boldsymbol{x} \in \{0,1\}^n\} = \mathrm{opt}\{P_{G^{\mathrm{ver}}}(\boldsymbol{x} \mid G^{\mathrm{sel}}) \mid \boldsymbol{x} \in \{0,1\}^n,\ \boldsymbol{x} \sim \chi_0^{\mathrm{ext}}\}$, where "$\chi_0^{\mathrm{ext}} \equiv \#$" and $G^{\mathrm{ver}} = G^{\mathrm{sel}} = G$.

When $\mathrm{opt}_{\boldsymbol{x} \in \{0,1\}^n,\ \boldsymbol{x} \sim \chi^{\mathrm{ext}}} P_{G^{\mathrm{ver}}}(\boldsymbol{x} \mid G^{\mathrm{sel}})$ needs to be calculated for some $G^{\mathrm{sel}}, G^{\mathrm{ver}} \subseteq G$, and an external coloring $\chi^{\mathrm{ext}} : V(G) \to C_0^{\mathrm{ext}} + C_1^{\mathrm{ext}} + \{\#\}$, we do the following:

1. If G^{ver} has size greater than some constant c, then find a $\sqrt{\cdot}$-separator for G^{ver} with $V(G^{\text{ver}}) = A_1 + S + A_2$.
2. For all internal colorings χ^{int} of S with $\chi^{\text{ext}} \rightsquigarrow \chi^{\text{int}}$ do:
   (a) Determine $\text{Eval}_{A_i}(\chi^{\text{int}})$ recursively for $i = 1, 2$.
   (b) Determine $\text{Eval}_S(\chi^{\text{int}})$.
3. Return $\text{opt}_{\chi^{\text{int}}, \chi^{\text{ext}} \rightsquigarrow \chi^{\text{int}}} h(\text{Eval}_{A_1}(\chi^{\text{int}}), \text{Eval}_S(\chi^{\text{int}}), \text{Eval}_{A_2}(\chi^{\text{int}}))$.

The size of the subproblems, i.e., the size of the graphs $G^{\text{ver}}_{A_i}(\chi^{\text{int}})$ which are used in the recursion, plays a crucial role in the analysis of the running time of this algorithm.

Definition 2. *A glueable select&verify problem is called* slim *if the subgraphs $G^{ver}_{A_i}(\chi^{int})$ are only by a constant number of vertices larger than $G^{ver}[A_i]$, i.e., if there exists a $\eta \geq 0$ such that $|V(G^{ver}_{A_i}(\chi^{int}))| \leq |A_i| + \eta$ for all internal colorings $\chi^{int} : S \to C^{int}$.*

Note that the proof of Lemma 1 shows that both VERTEX COVER and INDEPENDENT SET are slim with $\eta = 0$, whereas DOMINATING SET is not, as exhibited in the paper's long version. The following proposition gives the running time of the above algorithm. In order to assess the time required for the above given divide and conquer algorithm, we use the following abbreviations for the running times of certain subroutines: $T_S(n)$ denotes the time to find a separator in an n-vertex graph from class $\mathbb{G}$. $T_M(n)$ denotes the time to construct the modified graphs $G^{\text{ver}}_X(\chi^{\text{int}}) \in \mathbb{G}$ and the modified colorings $(\chi^{\text{ext}} \oplus_X \chi^{\text{int}})$ (for $X = \{A_1, S, A_2\}$ and each internal coloring χ^{int} with $\chi^{\text{ext}} \rightsquigarrow \chi^{\text{int}}$) from an n-vertex graph from class $\mathbb{G}$. $T_E(m)$ is the time to evaluate $\text{Eval}_S(\chi^{\text{int}})$ for all χ^{int}, $\chi^{\text{ext}} \rightsquigarrow \chi^{\text{int}}$, in a separator of size $m = \beta\sqrt{n}$. $T_g(n)$ is the time for gluing the results obtained by two sub-problems each of size $O(n)$. In the following, we assume that all these functions are polynomials.

Proposition 1. *For every $G \in \mathbb{G}$, $\text{opt}_{\boldsymbol{x} \in \{0,1\}^n} P_G(\boldsymbol{x})$ can be computed in time*

$$c(\alpha', \beta, \sigma)^{\sqrt{n}} q(n), \quad \textit{where} \quad c(\alpha', \beta, \sigma) = \sigma^{\beta/(1-\sqrt{\alpha'})}.$$

Here, $\alpha' = \alpha + \epsilon$ for any $\epsilon \in (0, 1-\alpha)$ and the running time analysis only holds for $n \geq n_0(\epsilon)$, and q is some polynomial. If, however, $\mathcal{G}$ is slim or the $\sqrt{\cdot}$-separator theorem yields cycle separators, then the running time for the computation is $c(\alpha, \beta, \sigma)^{\sqrt{n}} q(n)$, which then holds for all n.

Proof. (Sketch) Let $T(n)$ denote the time to compute $\text{opt}_{\boldsymbol{x} \in \{0,1\}^n,\, \boldsymbol{x} \sim \chi^{\text{ext}}} P_{G^{\text{ver}}}(\boldsymbol{x} \mid G^{\text{sel}})$ for a graph $G^{\text{ver}} = (V^{\text{ver}}, E^{\text{ver}})$ with $n = |V^{\text{ver}}|$ (where $\chi^{\text{ext}} : V(G) \to C_0^{\text{ext}} + C_1^{\text{ext}} + \{\#\}$ is some external coloring and $G^{\text{sel}}, G^{\text{ver}} \subseteq G$). In the case of the existence of a cycle separator theorem or if the problem is slim, the recurrence we have to solve in order to compute an upper bound on $T(n)$ then reads as follows:

$$T(n) \leq \sigma^{\beta\sqrt{n}} \cdot 2\, T(\alpha n) \cdot \underbrace{(T_M(n) + T_E(\beta\sqrt{n}) + T_g(\alpha n + \beta\sqrt{n}))}_{=: T_{M,E,g}(n)} + T_S(n).$$

The functions $T_{M,E,g}(n)$ and $T_S(n)$ are polynomials. The solution is given by $T(n) \leq \sigma^{(\beta/(1-\sqrt{\alpha}))\cdot\sqrt{n}} q(n)$ for some polynomial $q(\cdot)$. In the general case, from the definition of glueability, we have that the size of the two remaining subproblems to be solved recursively is $|V(G^{\mathrm{ver}}_{A_i}(\chi^{\mathrm{int}}))| \leq \alpha n + \beta\sqrt{n}$ for each χ^{int} that preserves χ^{ext}. Since with $\alpha' = \alpha + \epsilon$, for some $\epsilon \in (0, 1-\alpha)$, the inequality $\alpha n + \beta\sqrt{n} \leq \alpha' n$ holds for sufficiently large n, the result follows. □

4.2 How (Linear) Problem Kernels Help

Proposition 1 together with the existence of problem kernels yields:

Theorem 1. *Suppose that $\mathcal{G}$ admits a problem kernel of polynomial size $p(k)$ on $\mathbb{G}$ computable in time $T_K(n,k)$. Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$, for a graph $G \in \mathbb{G}$, in time*

$$c(\alpha', \beta, \sigma)^{\sqrt{p(k)}} q(k) + T_K(n,k), \quad \textit{where} \quad c(\alpha', \beta, \sigma) = \sigma^{\beta/(1-\sqrt{\alpha'})}, \qquad (2)$$

and $\alpha' = \alpha + \epsilon$ for any $\epsilon \in (0, 1-\alpha)$, holding only for $n \geq n_0(\epsilon)$, where $q(\cdot)$ is some polynomial. If, however, $\mathcal{G}$ is slim or the $\sqrt{\cdot}$-separator theorem yields cycle separators, then the running time for the computation is $c(\alpha, \beta, \sigma)^{\sqrt{p(k)}} q(k) + T_K(n,k)$, which then holds for all k. □

In particular, this means that for glueable select&verify problems for planar graphs that admit a *linear* problem kernel of size dk, we get an algorithm of running time $O\big(c(\alpha', \beta, \sigma, d)^{\sqrt{k}} q(k) + T_K(n,k)\big)$, where $c(\alpha', \beta, \sigma, d) = \sigma^{\beta\sqrt{d}/(1-\sqrt{\alpha'})}$.

Since VERTEX COVER is a slim problem, Theorem 1 yields a $c^{\sqrt{gk}}$-algorithm for $\mathbb{G}_g$, where $\mathbb{G}_g$ denotes the class of graphs of genus bounded by g, see [5].

4.3 Towards Avoiding (linear) Problem Kernels: The Core Concept

We are going to introduce the novel notion of *problem cores*, which is closely related to that of problem kernels, but seemingly "incomparable" and tailored towards *unweighted minimization* select&verify problems. The idea is to restrict (only) the size of the "selection space", while – unlike in the setting of problem kernels – the whole (possibly large) problem instance may be still used for "checking".

Definition 3. *Consider an unweighted select&verify minimization graph problem $\mathcal{G}$ specified by $(P, \min)$. A* corer of size $p(k)$ *is a polynomial time computable mapping $\phi : ((V,E),k) \mapsto V_c$ satisfying $|V_c| \leq p(k)$, and $\exists \boldsymbol{x} = (x_1, \ldots, x_{|V|}) \in \{0,1\}^{|V|}(P_G(\boldsymbol{x}) \leq k \wedge \{v_i \in V \mid x_i = 1\} \subseteq V_c)$. The set V_c is also called the* problem core *of $\mathcal{G}$. If $p(k) = ak$, we call ϕ a* linear *corer. In this case V_c is called a* factor-a problem core.

Note that weighted minimization problems could also be treated similarly at the expense of further technical complications. Having a problem core automatically makes a select&verify problem a "simple" one: For the problem core V_c, which can be computed in polynomial time, it is enough to check all k-element subsets, giving fixed parameter tractability. Stirling's formula yields:

Lemma 2. *If a select&verify problem $\mathcal{G}$ has a size ak problem kernel or if the core is a factor-a core, then there is a "$\min\{(ea)^k, 2^{ak}\}$-algorithm" for $\mathcal{G}$.* □

Even though there seems to be no *general* interrelation between problem kernels and cores, for our purposes, the different concepts can be interchanged:

Theorem 2. *Let G be an n-vertex-graph from a graph class $\mathbb{G}$ for which a $\sqrt{\cdot}$-separator theorem for* weighted graphs *is known with constants α, β. Suppose that $\mathcal{G}$ admits a corer of size $p(k)$, which can be computed in polynomial time $T_C(n)$. Then, there is an algorithm to decide $(G,k) \in \mathcal{G}$, for $G \in \mathbb{G}$, in time*

$$c(\alpha', \beta, \sigma)^{\sqrt{p(k)}} q(k) + T_C(n), \quad \text{where} \quad c(\alpha', \beta, \sigma) = \sigma^{\beta/(1-\sqrt{\alpha'})},$$

and $\alpha' = \alpha + \epsilon$ for any $\epsilon \in (0, 1-\alpha)$, holding only for $n \geq n_0(\epsilon)$. If, however, $\mathcal{G}$ is slim or the $\sqrt{\cdot}$-separator theorem yields cycle separators, then the time for the computation is $c(\alpha, \beta, \sigma)^{\sqrt{p(k)}} q(k) + T_C(n)$, which holds for all k.

Proof. (Sketch) Consider $G = (V, E) \in \mathbb{G}$. The algorithm proceeds as follows:
(1) Compute a core $V_c \subseteq V$ containing at most $p(k)$ vertices in time $T_C(n)$.
(2) Then, $G[V_c] \in \mathbb{G}$ is the graph from which vertices have to be selected.
(3) Find an optimal $\boldsymbol{x}$ satisfying $P_G(\boldsymbol{x} \mid G[V_c])$ by applying the algorithm outlined before Prop. 1. One modification of this algorithm, however, is necessary: we do not use a separator theorem in step 1 of that algorithm for a separation of G^{ver}, but we add *weights* to G^{ver} as follows: V_c induces a weight function ω on G^{ver} by letting $\omega(v) = 1/|V_c|$, if $v \in V_c$ and $\omega(v) = 0$, otherwise. Then, we apply a separator theorem for weighted graphs to G^{ver} with the weight function ω. The constants $c(\alpha, \beta, \sigma)$ and $c(\alpha', \beta, \sigma)$ then are derived as in Prop. 1. □

5 Conclusion and Further Results

Summary. We exhibited how to use separator theorems for obtaining $c^{\sqrt{k}}$-fixed parameter divide and conquer algorithms. We defined "glueable select&verify problems," capturing graph problems such as VERTEX COVER and DOMINATING SET, as a problem class that allows for a divide and conquer approach on certain graph classes. Admittedly, the constants within these algorithms are still rather huge. For example, in the case of PLANAR VERTEX COVER, Theorem 1 yields a $37181^{\sqrt{k}}$-algorithm. In the long version, we elaborate on several ideas to overcome this weakness:

Further results (see the full version of the paper). Firstly, we analyze how Lipton and Tarjan proved their famous planar separator theorem; basically, the proof consists of two steps: in the first step, the given graph is thinned into pieces of "small" radius, and in the second step, a special (cycle) separator lemma for planar graphs with bounded radius is used. The such obtained separator therefore consists of two parts. Since after one application of Lipton and Tarjan's separator theorem (in a divide and conquer algorithm as described in Section 4),

the remaining graph pieces still have relatively small radius, one could avoid the first thinning step. Iterating this idea in an optimized fashion, one gets, e.g., in the case of PLANAR VERTEX COVER, a $8564^{\sqrt{k}}$-algorithm. It is a challenge if better algorithm bounds are obtainable by using other separator theorems.

Secondly, we discuss the idea of stopping the recursion before having graph parts of constant size and then applying, e.g., elaborated search tree algorithms to these small parts. More precisely, a divide and conquer algorithm would once use Lipton and Tarjan's planar separator theorem for slicing the graph into pieces of small radius and then use only the mentioned special (cycle) separator lemma in the remainder of the recursion, until all graph pieces are sufficiently small. In this way, one gets $c^{k^{2/3}}$-algorithms with reasonable small constants c. For example, we derive a $7.7670^{k^{2/3}}$-algorithm for PLANAR VERTEX COVER.

Future research. We briefly sketch three lines of future research. (1) An alternative idea in order to lower the involved constants would be to devise new separator theorems with constants α and β, not only concentrating on bringing down β for fixed α (as, e.g., done for $\alpha = 2/3$ in [6]), but on minimizing the function $\beta/(1-\sqrt{\alpha})$. (2) It is an issue of future research to further investigate the newly introduced concept of cores. For example: is there a linear size core (or kernel) for PLANAR DOMINATING SET? (3) Finally, this paper (together with [1,2]) might be a stimulus to investigate the structure "within" FPT in order to distinguish problems allowing for $c^{\sqrt{k}}$-algorithms from problems which seem to allow only for c^k-algorithms. Is it possible to develop a reasonable fine-grained structural theory of FPT? Note that Cai and Juedes [3] very recently showed that for a list of parameterized problems (e.g., for VERTEX COVER on general graphs) $c^{o(k)}$-algorithms are impossible unless FPT = $W[1]$.

References

1. J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proc. 7th SWAT*, vol. 1851 of *LNCS*, Springer, pp. 97–110, 2000.
2. J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. Technical Report TR01–023, ECCC Reports, Trier, March 2001. Extended abstract in *Proc. 28th ICALP*, Springer, 2001.
3. L. Cai and D. Juedes. Subexponential parameterized algorithms collapse the W-hierarchy. In *Proc. 28th ICALP*, Springer, 2001.
4. J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. In *Proc. 25th WG*, vol. 1665 of *LNCS*, Springer, pp. 313–324, 1999.
5. H. N. Djidjev. A separator theorem for graphs of fixed genus. *Serdica*, 11:319–329, 1985.
6. H. N. Djidjev and S. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.
7. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
8. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.
9. R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

On Assigning Prefix Free Codes to the Vertices of a Graph

N.S. Narayanaswamy*[1] and C.E. Veni Madhavan[2]

[1] Institut für Informatik, Ludwig Maximilians Universität,
Oettingenstrasse 67, 80538 Munich, Germany,
swamy@informatik.uni-muenchen.de.
[2] Department of Computer Science and Automation,
Indian Institute of Science, Bangalore 560 012, India,
cevm@csa.iisc.ernet.in.

Abstract. For a graph G on n vertices, with positive integer weights $w_1, \ldots, w_n$ assigned to the n vertices such that, for every clique K of G, $\sum_{i \in K} \frac{1}{2^{w_i}} \leq 1$, the problem we are interested in is to assign binary codes $C_1, \ldots, C_n$ to the vertices such that C_i has w_i (or a function of w_i) bits in it and, for every edge $\{i, j\}$, C_i and C_j are not prefixes of each other. We call this the *Graph Prefix Free Code Assignment Problem.* We relate this new problem to the problem of designing adversaries for comparison based sorting algorithms. We show that the decision version of this problem is as hard as graph colouring and then present results on the existence of these codes for prefect graphs and its subclasses.

1 Introduction

Our study of the Graph Prefix Code Assignment problem is motivated by the question of designing adversaries for *sorting under partial information* (discussed below). This motivation was suggested by Subrahmanyam and Dubhashi [1].

The problem of *sorting under partial information* is: Given a partial order P on a set $X = \{x_1, \ldots, x_n\}$, determine a hidden linear extension of P using questions of the form *is* $x_i < x_j$*?*. An algorithm to solve this problem is said to sort P by comparisons. It is also referred to as a comparison based algorithm for sorting. In this paper we only consider comparison based sorting algorithms.

A comparison based sorting algorithm can be seen as a game between two players D and H. In this game D has a partial order P and H holds a linear extension of P, T, hidden from D. D tries to discover T by asking questions about the relation in T (comparisons) between pairs of elements. D has to make $\log e(P)$ comparisons to discover T where, $e(P)$ is the number of linear extensions of P. An interesting question here is the issue of designing an adversary who answers on-line, the comparison queries of an algorithm, so as to force the algorithm to use $\log e(P)$ comparisons. This problem was solved by Brodal et

* Supported by DFG Grant No. Jo 291/2-1

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 328–337, 2001.
© Springer-Verlag Berlin Heidelberg 2001

al. in [7] for the case when P is the anti-chain on X. They design a very simple and elegant adversary that answers the comparison queries of any sorting algorithm to force it to use $\Omega(\log n!)$ comparisons. For the general case when there is no restriction on P, such an adversary was designed by Kim and Kahn in [2] using a relation they prove between $e(P)$ and the entropy of the comparability graph of P(defined below). The adversary answers each comparison query depending on the outcome of an entropy calculation (depending on P and answers to past queries). A question that remains open in this area, posed in [1], is to design an adversary for sorting under partial information that avoids the entropy calculations that are an integral part of the adversary designed in [2].

In this paper we present the relation between the design of adversaries for sorting under partial information and the Graph Prefix free Codes assignment problem. We present the first results on the prefix free codes problem. Our results are stated after the preliminaries in section 2.

2 Preliminaries and Our Results

A collection of positive integers $w_1, \ldots, w_r$ is said to satisfy *Kraft's Inequality* if $\sum_{i=1}^{r} \frac{1}{2^{w_i}} \leq 1$.

Throughout the paper, G is an undirected graph with positive integer weights $w_1, \ldots, w_n$ assigned to its vertices $v_1, \ldots, v_n$. The weights satisfy the property that for every clique in G, the weights assigned to its vertices satisfy Kraft's inequality. The weights assigned to the vertices of a graph G are said to satisfy Kraft's inequality if the induced weight assignment to every clique in G satisfies Kraft's inequality. Unless mentioned otherwise the weight assignment $\langle w_1, \ldots, w_n \rangle$ to vertices $\langle v_1, \ldots, v_n \rangle$ implies v_i is assigned the weight w_i. A similar convention holds for the code assignment $\langle C_1, \ldots, C_n \rangle$. Two codes are prefix free if neither is a prefix of the other.

Prefix Free Code Assignment Problem for Graphs: For a given positive integer c, a graph G with weights $w_1, \ldots, w_n$ assigned to vertices $v_1, \ldots, v_n$ satisfying Kraft's inequality, does there exist an assignment of binary codes $C_1, \ldots, C_n$ to the vertices $v_1, \ldots, v_n$ such that number of bits in C_i is less than or equal to $w_i + c$, and for every edge $\{i, j\}$ in G, and such that C_i and C_j are not prefixes of each other?

We call such a code a *$+c$ prefix free code* for $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$. We use $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$ to denote G with the weights $\langle w_1, \ldots, w_n \rangle$ assigned to its vertices.

Prefix Free Codes and Tree Embeddings: Here we present the relation between prefix free codes and a partial order. Let T denote the infinite binary tree. With each node of T, associate a binary string in a natural way: the root is associated with the empty string, and recursively, if a node is associated with a string w, its children are associated with the sequences $w0$ and $w1$. T can

be seen as a partial order with the elements related by the ancestor-descendant relationship. Consequently, it follows that two codes are prefix free if and only if they form an anti-chain in T. Therefore, our question boils down to assigning vertex $v_i, i \in \{1, \ldots, n\}$, to a node in the tree T at level $w_i + c$ or lesser such that the nodes assigned to adjacent vertices in the graph form an anti-chain in T. This problem is now a natural variant of the *Graph Vertex Colouring* problem. In the Graph Colouring problem, the colours or codes to be assigned to the vertices come from the partial order in which any two distinct elements are unrelated. Therefore, two adjacent vertices must be given colours that are unrelated in this partial order. On the other hand, in the Prefix Free code assignment problem, we are given a tree partial order, and weights to the vertices. The code assignment criterion is that two adjacent vertices get unrelated codes in the partial order and, there is a restriction on the layers where the codes come from.

Definitions Related to Partial Orders and Graph Entropy: The *comparability graph*, $G(P)$, of a partial order P on a set $X = \{x_1, \ldots, x_n\}$, is an undirected graph with X as its vertex set, and $\{x_i, x_j\} \in E(G(P))$ iff x_i and x_j are comparable in P.

We write $P(x < y)$ to denote the partial order obtained from P by adding the relation $x < y$. We refer only to a partial order without referring to the set on which it is defined when there is no confusion about the underlying set.

The *vertex packing polytope*, $VP(G)$, of a graph G with V as a vertex set, is the convex hull of the set of indicator vectors of stable (independent) sets of V. Recall that the indicator vector, I_A, of a set $A \subseteq V$, is a $|V|$-dimensional vector in which $(I_A)_i = 1$ if the i-th vertex of V is in A. Otherwise, it is zero.

The *graph entropy* of G with respect to a probability distribution p on $V(G)$ is

$$H(G, p) = \min_{a \in VP(G)} - \sum_{i=1}^{n=|V|} p_i \log a_i$$

In this paper we are interested only in the case when p is the uniform distribution on the set of vertices and therfore, entropy is denoted by $H(G)$. The entropy of the comparability graph of a partial order P is denoted by $H(P)$. We will use the phrase entropy of a partial order P to mean the entropy of the comparability graph of P.

2.1 Our Results

We first present the relation between the prefix free codes problem and the problem of designing adversaries for sorting under partial information. Using the similarity of the prefix free code assignment problem to graph colouring, we show that

- For an input $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$ where $w_1, \ldots, w_n$ satisfy the Kraft's inequality, there is no polynomial time algorithm to decide if there is a $+0$-prefix free code unless $P = NP$.

We address the problem of the existence of these codes and prove that

– For every positive integer c, there is a graph G, and a vertex weight assignment $\langle w_1, \ldots, w_n \rangle$ such that there is no $+c$-prefix free code.

We then observe that the graphs we construct in the above proof are imperfect graphs, that is, graphs for which there is an induced subgraph whose clique number and chromatic number are different. So we concentrate our exploration on perfect graphs. This direction is also consistent with our goal of designing adversaries for sorting where, the graphs that arise are comparability graphs (this class of graphs are known to be perfect).

– We first construct a perfect graph along with vertex weights for which there is no $+0$-prefix free code. We then show that for any perfect graph with weights $\langle w_1, \ldots, w_n \rangle$ satisfying Kraft's inequality, there is a prefix free code assignment whose code lengths are within a multiplicative factor of 2 from the specified weights.

We then investigate the problem for specific classes of perfect graphs namely chordal graphs. Specifically,

– we construct a chordal graph G and associated vertex weights $w_1, \ldots, w_n$ such that $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$ does not have a $+1$-prefix free code.

Plan of the Paper. In Section 3 we present the relation between designing adversaries for sorting under partial information and the prefix free code assignment problem. In Section 4 we present the hardness result for the prefix free code assignment problem. In Section 4.1 we prove non existence results for $+c$-prefix free codes for general graphs. In Section 4.2 we study the problem in relation to the perfect graphs. Section 5 presents our results on chordal graphs.

3 Motivation for the Prefix Free Code Assignment Problem

In this section we present the connection to the problem of designing adversaries for sorting under partial information. We first present the adversaries due to Brodal [7] and Kim and Kahn[2]. We then point out a method for designing an adversary, which leads to the prefix free code assignment problem. To make the distinction between the two adversaries, we refer to the adversary of Brodal as the *adversary for sorting* and the adversary of Kim and Kahn as the *adversary for sorting under partial information*.

3.1 Adversary for Sorting

The adversary that we present is one that forces any sorting algorithm to use $\Omega(\log n!)$ comparisons.

The Adversary. We describe the adversary of [7] for answering comparisons made by a sorting algorithm. At the outset (before any comparisons are made), the adversary places all the elements of X at the root of an infinite binary tree. For $x \in X$, let $v(x)$ denote the node of the tree to which x has been assigned. The adversary answers a comparison between elements $x, y \in X$ based on the follows cases:

- $v(x) = v(y)$: Then x is moved to the left child of $v(x)$ and y to the right child and the adversary answers $x < y$.
- $v(x)$ is a descendant of $v(y)$: y is moved to the unique child of $v(y)$ that is not an ancestor of $v(x)$. If this child is the left child then the adversary answers $y < x$ and if it is a right child it answers $x < y$.
- $v(x)$ and $v(y)$ are unrelated: If $v(x)$ is visited before $v(y)$ in a pre-order traversal of the tree, the adversary answers $x < y$ and otherwise the adversary answers $y < x$.

It is proved in [7] that this adversary forces any sorting algorithm to use at least $\frac{n \log n}{2}$ comparisons. The key ingredients in the proof of this claim are the following:

- In the worst case, each comparison pushes two elements down by one level in the tree.
- At the end of the algorithm, each node of the tree will contain at most one element.
- Sum of the distances from the root to nodes that contain an element of X is at least $n \log n$ and this is equal to sum of the number of levels each element has been pushed down. This statement is the most important ingredient in the proof of the claim.
- Each comparison contributes at most two to the sum in the previous item. Hence, the number of comparisons made is at least $\frac{n \log n}{2}$.

3.2 Adversary for Sorting under Partial Information

Here we describe the adversary for algorithms that discover a linear extension of a partial order P that is given to them. The adversary described below answers the comparison queries on-line, in a manner so as to force the algorithm to make at least $\log e(P)$ comparisons. The adversary is based on the following two theorems proved in [2] which relate the number of linear extensions of P and $H(P)$.

Theorem 1. *For any partial order P,*

$$n(\log n - H(P)) \geq \log e(P),$$

$$\log e(P) \geq max\{\log n! - nH(P), Cn(\log n - H(P))\},$$

where $C = \frac{1}{1+7\log e}$

Theorem 2. *For any partial order P and any incomparable elements $x, y \in P$,*

$$min\{H(P(x < y)), H(P(y < x))\} \leq H(P) + \frac{2}{n}.$$

The Adversary. We describe the answers the adversary gives to a comparison query made by an algorithm that knows a partial order P. The adversary answers a comparison between elements x, y based on the follows cases:

- x and y are comparable in P: The adversary answers exactly the relation between x and y in P.
- x and y are incomparable in P: The adversary applies theorem 2. The adversary answers $x < y$ if $H(P(x < y)) = min\{H(P(x < y)), H(P(y < x))$. Otherwise the adversary answers $y < x$.

In the first case, after getting the answer from the adversary, the algorithm still knows only P. Whereas, in the second case, the algorithm knows a new partial order which is either $P(x < y)$ or $P(y < x)$ depending on the answer given by the adversary. It is proved in [2] that this adversary forces any algorithm for sorting under partial information to use $\Omega(\log e(P))$ comparisons. The key ingredients in the proof of this claim are the following:

- The entropy of the comparability graph of a total order is $\log n$. This is because the comparability graph of a total order is a clique.
- By theorem 2, each comparison yields a new partial order whose entropy is at most $\frac{2}{n}$ more than the entropy of the current partial order.
- The above two observations yield a lower bound on the number of comparisons, namely, $(\log n - H(P))\frac{n}{2}$. From theorem 1 this is at least $\frac{\log e(P)}{2}$.

3.3 A Simpler Adversary for Sorting Partial Orders and the Prefix Free Code Assignment Problem

We now describe a new adversary for sorting under partial information. This adversary starts by placing the elements of X at nodes in the infinite binary tree. The placement is based on the partial order P. The answers to the comparison queries are given using the rules of the adversary for sorting. We state the following theorem that describes the placement of the elements of X in the tree:

Theorem 3. *Let P be a partial order on a set $X, |X| = n$. Let $(a_1, \ldots, a_n)$ be a point in $VP(G(P))$ such that $H(G(P)) = \sum_{i=1}^{n} -\log a_i$. Let every element of X be assigned to a corresponding node of the infinite binary tree satisfying the following properties (tree embedding condition):*

- *For every $x, y \in X$, if x and y are comparable in P, then the nodes in the tree assigned to x and y are unrelated in the tree partial order.*
- *Element $i \in X$ is assigned to a node in the infinite binary tree at distance $\lceil -\log a_i \rceil$ from the root.*

Consider an adversary who:

- *Starts with the above placement of elements in the infinite binary tree.*
- *Answers the comparison queries of any algorithm sorting P using the rules of the adversary for sorting.*

This adversary will force any algorithm for sorting under partial information to use at least $\frac{\log e(P)-n}{2}$ *comparisons.*

Proof. As in the analysis of the adversary for sorting, when the sorting algorithm ends, the adversary would have placed each element of X to a tree node bijectively. The sum of distances from the root to nodes containing elements of X is $n \log n$. The distance from root of an element finally consists of two components: its distance from the root in the tree configuration the adversary starts with, and the distance increase due to comparisons. For each comparison, the contribution to the sum of distances increases by at most 2. Therefore, it follows that $n \log n \leq \sum_{i=1}^{n} \lceil -\log a_i \rceil + 2\times$ number of comparisons. Using the fact that $H(P) = \frac{1}{n} \sum_{i=1}^{n} -\log a_i$, simple arithmetic and theorem 1 it follows that number of comparisons is at least $\frac{\log e(P)-n}{2}$.

This lower bound on the number of comparisons is of significance for partial orders for which $e(P)$ is large.

Remarks on the Tree Embedding Condition in the Theorem: The condition in the theorem requiring that two related elements in P be assigned two unrelated nodes in the tree partial order is same as asking for the codes assigned to them to be prefix free. The question that is of interest is, *Does the tree embedding condition hold for every partial order P with respect to the point in $VP(G(P))$ that achieves $H(P)$?* Furthermore, there is no specific reason as to why the tree embedding condition should hold only with respect to a point at which entropy of P is achieved.

Also, from [3] it is true that $VP(G(P))$ consists of exactly those vectors $(v_1, \ldots, v_n) \in [0,1]^n$ for which $\sum_{i \in I} v_i \leq 1$ whenever $\{x_i \in X | i \in I\}$ is a clique in $G(P)$. This statement is true for all perfect graphs and $G(P)$ is a perfect graph. The above remarks lead to the following problem:

Let P be a partial order on a set of n elements. Let $w_1, \ldots, w_n$ be positive integer weights assigned to the vertices of $G(P)$ such that $(\frac{1}{2^{w_1}}, \ldots, \frac{1}{2^{w_n}}) \in VP(G(P))$. Do there exist binary codes $C_1, \ldots, C_n$ to the vertices such that C_i has w_i bits in it and, for every edge $\{i, j\} \in G(P)$, C_i and C_j are not prefixes of each other?

The above problem can be asked for any undirected graph. For an undirected graph there is a simple necessary condition for the existence of prefix free codes: the weights $(w_1, \ldots, w_n)$ should satisfy the Kraft's inequality. If the weights satisfy Kraft's inequality then $(\frac{1}{2^{w_1}}, \ldots, \frac{1}{2^{w_n}}) \in VP(G(P))$ [3]. Therefore, for arbitrary graphs we get the Graph Prefix Free Code assignment problem:

Let G be a graph on n vertices, with positive integer weights $w_1, \ldots, w_n$ assigned to the n vertices such that the Kraft's inequality is satisfied. Is there an assignment of binary codes $C_1, \ldots, C_n$ to the vertices such that C_i has w_i bits in it and, for every edge $\{i, j\}$, C_i and C_j are not prefixes of each other?

4 Hardness and Existence Results

In this section, we show that the there is no polynomial time algorithm to decide if $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$ has a +0-prefix free code assignment unless $P = NP$.

Theorem 4. *Given a graph G, and a weight assignment $\langle w_1, \ldots, w_n \rangle$ that satisfies the Kraft's inequality, there is no polynomial (in the size of the graph) time algorithm to decide if $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$ has a +0-prefix free code, unless $P = NP$.*

Proof. The proof is by showing that a polynomial time algorithm to decide the existense of a +0-prefix free code can be used to obtain in polynomial time algorithm for an NP-complete problem. The problem we consider is that of finding in polynomial time a constant factor, less than 2, approximation to the minimum colouring of an undirected graph. Our approximation algorithm would find the smallest $i, 1 \leq i \leq \log n$ such that $\langle G, \langle i, \ldots, i \rangle \rangle$ has a +0-prefix free code. This prefix free code assignment would be a 2^i colouring of G. Since i is the smallest value for which there is a 2^i colouring, this gives a constant factor, less than 2, polynomial time approximation algorithm.

4.1 Existence Results

For every positive integer c, we observe that there is a graph G and a vertex weight assignment such that $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$ does not have a $+c$ prefix free code.

Theorem 5. *For every positive integer c, there is a graph G with vertex weights $\langle w_1, \ldots, w_n \rangle$ that satisfies Kraft's inequality, and $\langle G, \langle w_1, \ldots, w_n \rangle \rangle$ has no $+c$-prefix free code.*

Proof. We observe that there exist graphs in which the largest clique is an edge, but the chromatic number is arbitrarily large. In particular, for any positive integer c, let G_c be a graph whose largest clique is an edge and the chromatic number (minimum colouring number) is 2^{c+2}. Let the weight assignment of G_c be $\langle 1, \ldots, 1 \rangle$. This weight assignment satisfies Kraft's inequality in G_c. Clearly, G does not have a $+c$ prefix free code assignment.

The construction of graphs with a fixed largest clique number and arbitrarily large chromatic number was first given by Lovasz. A construction of such a graph is presented in the book by Bondy and Murthy [4]. Clearly, the graph that we use in the construction is an *imperfect graph*, that is graphs which have an induced subgraph whose chromatic and clique number are different. We next consider the prefix free code assignment problem for the case when the graphs are *perfect* (meaning, not imperfect). The exploration on this class of graphs is well motivated given that comparability graphs are perfect graphs.

4.2 Prefix Free Codes for Perfect Graphs

We next show that the perfectness of a graph is insufficient for +0-prefix free codes to exist.

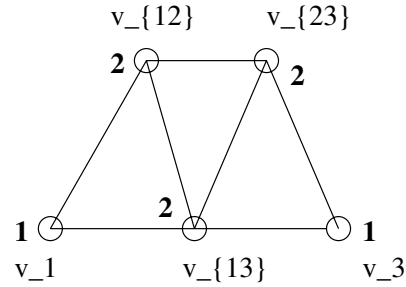


Fig. 1. This graph doesn't have a +0 prefix free code for the shown weights.

A Perfect Graph with no +0-Prefix Free Code: The graph we present here is an Interval Graph. Interval graphs are intersection graphs of intervals on a line and are known to be perfect [5]. The graph I that we construct is the graph in Figure 1. It can be shown that $\langle I, \langle 1, 2, 2, 2, 1\rangle\rangle$ does not have a +0-prefix free code. This construction shows that there exist perfect graphs and weight assignments that do not have a +0-prefix free code. We improve on this construction and show in the Section 5 a similar result for a sub class of perfect graphs namely *chordal graphs*. Now we use the fact that the colouring problem can be solved in polynomial time for perfect graphs, to assign a prefix free code whose lengths are within a multiplicative factor of 2.

Prefix Free Codes within a Constant Factor of the Weight for Perfect Graphs Here we show that the satisfaction of Kraft's inequality by the weights $\langle w_1, \ldots, w_n\rangle$ for a perfect graph G is sufficient for a prefix free code $\langle C_1, \ldots, C_n\rangle$ to exist such that the number of bits in C_i is at most $2w_i$.

Theorem 6. *Let G be a perfect graph with vertex weights $\langle w_1, \ldots, w_n\rangle$ that satisfy Kraft's inequality. G has a prefix free code $\langle C_1, \ldots, C_n\rangle$ such that the number of bits in C_i is at most $2w_i$. Moreover, such an assignment of prefix free codes can be computed in polynomial time.*

Proof. We prove the theorem by constructing a prefix free code. The construction is as follows: Let there be r distinct weights such that $w_1 < \ldots < w_r$ among the weights $\langle w_1, \ldots, w_n\rangle$. Consider the r induced subgraphs of G, denoted $G_1, \ldots, G_r$. G_i consists of vertices whose weight is w_i in G. By definition, each of the graphs $G_1, \ldots, G_r$ is perfect. We find an optimal colouring in polynomial time for each G_i. Due to the perfectness of G_i and the Kraft's inequality, the optimal colouring of G_i uses at most 2^{w_i} colours. This colouring gives a prefix free code for each G_i in which every code has w_i bits. Now for each i, we prefix the codes given to the vertices in G_i by $i-1$ 1s followed by a 0. As $i \leq w_i$, a vertex of weight w_i gets a code of length at most $2w_i$. Clearly, the code assignment is prefix free. The above construction can be run in polynomial time using the polynomial time algorithm for perfect graph colouring [3]. Hence the theorem.

5 Prefix Free Codes for Chordal Graphs

We construct a chordal graph and a weight assignment for which there is no +1-prefix free code. This construction can be extended to give a chordal graph which has no $+c$-prefix free code for any positive integer c. We present the construction of the graph in a step by step manner.

- Start with a clique on 2^w vertices, with each vertex of weight w. Clearly this weight assignment satisfies Kraft's inequality. Let G_0 denote this graph.
- For each sub-clique K_1 of G_0 with $2^{\frac{w}{2}}$ vertices, add a clique K_2 with $2^{\frac{w}{2}} - 1$ vertices, each of weight $\frac{w}{2}$. Each vertex in K_1 is adjacent to every vertex in K_2 that is, $K_1 \bigcup K_2$ is a clique. Let G_1 denote the resulting graph. The weight assignment to $K_1 \bigcup K_2$ satisfies Kraft's inequality.
- For each sub-clique K_1 of G_1, that has $2^{\frac{w}{4}+1}$ vertices such that $2^{\frac{w}{4}}$ of them have weight w and the rest have weight $\frac{w}{2}$, add a clique K_2 with $2^{\frac{w}{4}}(1 - 2^{\frac{w}{4}}(\frac{1}{2^w} + \frac{1}{2^{\frac{w}{2}}}))$ vertices, each of weight $\frac{w}{4}$. As before, $K_1 \bigcup K_2$ forms a clique and the weight assignment satisfies Kraft's inequality. Let G denote the resulting graph.

Theorem 7. *G is a chordal graph, and given the weight assignment as mentioned above for $w \geq 8$, there is no +1-prefix free code.*

Proof. The proof is omitted due to lack of space.

Summary We have motivated the graph prefix free coding problem as a natural problem that arises from the issue of designing adversaries for sorting under partial information. The cases of the problem when the graphs are comparability graphs or interval graphs are of specific interest. Positive results in the comparability graph case would yield adversaries for sorting under partial information and positive results in the interval graph case could be applied to the Dynamic Storage Allocation problem [6][$p226, SR2$].

References

1. Devdutt Dubashi, and K.V. Subrahmanyam, Notes on Prefix-free codes, Entropy and Sorting. *Personal Communication*
2. Jeff Kahn, and Jeong Han Kim, Entropy and Sorting. *24th Annual ACM STOC, pp 178-187, 1992*
3. Martin Grostshcel, Laszlo Lovasz, Alexander Schrijver, Geometric Algorithms and Combinatorial Optimization. *Springer Verlag*
4. J.A. Bondy, and U.S.R. Murthy Graph Theory with Applications. *Elsevier North-Holland 1976*
5. Martin C. Golumbic, Algorithmic Graph Theory and Perfect Graphs. *Academic Press, 1980*
6. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* (Freeman, San Fransisco, CA, 19 79)
7. G.S. Brodal, Shiva Chaudhuri, J. Radhakrishnan, The Randomized Complexity of Maintaining the Minimum. *Nordic Journal of Computing, Selected Papers of the 5th Scandinavian Workshop on Algorithm Theory (SWAT'96), 3(4):337–351, 1996.*

A New Measure of Edit Distance between Labeled Trees*

Chin Lung Lu[1], Zheng-Yao Su[1], and Chuan Yi Tang[2]

[1] National Center for High-Performance Computing,
P.O. Box 19-136, Hsinchu, Taiwan 300, R.O.C.
{cllu,zsu}@nchc.gov.tw
[2] Department of Computer Science, National Tsing Hua University,
Hsinchu, Taiwan 300, R.O.C.
cytang@cs.nthu.edu.tw

Abstract. One of the most important problem in computational biology is the tree editing problem which is to determine the edit distance between two rooted labeled trees. It has been shown to have significant applications in both RNA secondary structures and evolutionary trees. Another viewpoint of considering this problem is to find an edit mapping with the minimum cost. By restricting the type of mapping, Zhang [7,8] and Richter [5] independently introduced the constrained edit distance and the structure respecting distance, respectively. They are, in fact, the same concept. In this paper, we define a new measure of the edit distance between two rooted labeled trees, called less-constrained edit distance, by relaxing the restriction of constrained edit mapping. Then we study the algorithmic complexities of computing the less-constrained edit distance between two rooted labeled trees. For unordered labeled trees, we show that this problem is NP-complete and even has no absolute approximation algorithm unless P = NP, which also implies that it is impossible to have a PTAS for the problem. For ordered labeled trees, we give a polynomial-time algorithm to solve the problem.

1 Introduction

A *labeled tree* is a tree whose nodes are labeled from a finite alphabet Σ. A *rooted* tree is a tree in which one of the vertices is designated as *root*. An *unordered* labeled tree is just a rooted labeled tree. An *ordered* labeled tree is a rooted labeled tree in which the children of each node are ordered, i.e., if a node has k children, then we can specify them as the first child, the second child, $\cdots$, and the kth child. Unless otherwise stated, all trees we consider in this paper are rooted labeled trees.

Let λ be a symbol not in Σ and $a, b \in \Sigma \cup \{\lambda\}$. We use $a \to b$ to denote an edit operation. Given a tree T, we consider three kinds of edit operations on T. (1) *Change* (i.e., $a, b \in \Sigma$): means to change a node a to b. (2) *Deletion* (i.e.,

* Supported partly by the National Science Council of the Republic of China under grant NSC89-2213-E-321-004.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 338–348, 2001.
© Springer-Verlag Berlin Heidelberg 2001

$b = \lambda$): means to delete a node a such that the children of a become the children of the parent of a. (3) *Insertion* (i.e., $a = \lambda$): means to insert a node b as a child of some one node, say c, such that b becomes the parent of some (consecutive, if T is ordered) children of c.

Let γ be a cost function which assigns each edit operation $a \to b$ a non-negative real number $\gamma(a \to b)$ such that (1) $\gamma(a \to a) = 0$, (2) $\gamma(a \to b) = \gamma(b \to a)$, and (3) $\gamma(a \to c) \leq \gamma(a \to b) + \gamma(b \to c)$. The above function γ is called a *metric* cost function. Let $S = (s_1, s_2, \cdots, s_k)$ be a sequence of edit operations. S is called an *edit sequence* of transforming tree A to tree B if there are trees $T_1, T_2, \cdots, T_{k+1}$ such that $A = T_1$, $B = T_{k+1}$ and T_{i+1}, where $1 \leq i \leq k$, is obtained by applying s_i to T_i. The *cost* of S is defined to be $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$.

For any two trees T_1 and T_2, the *edit distance* between them, denoted by $D_e(T_1, T_2)$, is defined to be the minimum cost of an edit sequence transforming T_1 to T_2. That is, $D_e(T_1, T_2) = \min_S\{\gamma(S) | S \text{ is an edit sequence transforming } T_1 \text{ to } T_2\}$. The problem of computing $D_e(T_1, T_2)$ can be considered from a graphical viewpoint, called edit mapping. Let $V(T_i)$ be the set of nodes of tree T_i. An *edit mapping* from T_1 to T_2, denoted by $M_e(T_1, T_2)$, is defined to be a subset of $V(T_1) \times V(T_2)$ such that for any two (u_1, v_1) and (u_2, v_2) in $M_e(T_1, T_2)$, the following conditions are satisfied:

1. $u_1 = u_2$ if and only if $v_1 = v_2$ (one-to-one mapping).
2. u_1 is an ancestor of u_2 if and only if v_1 is an ancestor of v_2 (i.e., the ancestor-descendant relationship is preserved).
3. (Only for ordered labeled trees) u_1 is to the left of u_2 if and only if v_1 is to the left of v_2 (i.e., the sibling order is preserved).

See Figure 1 for an example of an edit mapping between two ordered labeled trees T_1 and T_2. We simply use M_e instead of $M_e(T_1, T_2)$ if there is no confusion. We call a pair $(u, v) \in M_e$ a *mapped line* and say that u and v are *covered* by (u, v) or M_e. In fact, the edit mapping shows a way to transform T_1 to T_2 as follows. Each mapped line (u, v) indicates that u should be changed to v if $u \neq v$; otherwise, u remains unchanged. The uncovered nodes of T_1 are to be deleted and the uncovered nodes of T_2 are to be inserted. For example, the edit mapping of Figure 1 corresponds to an edit sequence $(c \to \lambda, f \to g, \lambda \to c)$ to transform T_1 to T_2.

Let U and V be the sets of the uncovered nodes in T_1 and T_2, respectively. Then the *cost* of M_e is defined to be $\gamma(M_e) = \sum_{(u,v) \in M_e} \gamma(u \to v) + \sum_{u \in U} \gamma(u \to \lambda) + \sum_{v \in V} \gamma(\lambda \to v)$. In [10], Zhang and Shasha showed that the edit sequence with the minimum cost corresponds to an edit mapping with the minimum cost. In other words, the edit distance between T_1 and T_2 can be restated as $D_e(T_1, T_2) = \min\{\gamma(M_e) | M_e \text{ is an edit mapping between } T_1 \text{ and } T_2\}$.

Let $lca(u, v)$ represent the least common ancestor of nodes u and v in a tree. Motivated by [6], Zhang [7,8] introduced the *constrained edit mapping*, denoted by M_c, which is an edit mapping satisfying an additional condition that for any triple $(u_1, v_1), (u_2, v_2)$ and (u_3, v_3) in M_c, $lca(u_1, u_2)$ is a proper ancestor of u_3 if and only if $lca(v_1, v_2)$ is a proper ancestor of v_3. In [5], Richter defined the

structure respecting mapping, denoted by M_s, which is an edit mapping satisfying an additional condition that for any triple $(u_1, v_1), (u_2, v_2)$ and (u_3, v_3) in M_s such that none of u_1, u_2 and u_3 is an ancestor of one of the others, $lca(u_1, u_2) = lca(u_1, u_3)$ if and only if $lca(v_1, v_2) = lca(v_1, v_3)$. In fact, both the concepts of constrained edit mapping and structure respecting mapping are equivalent. Each one is a restricted variant of the edit mapping which prohibits the edit mappings as showing in Figures 3 and 4. The mapping of Figure 2 is a constrained edit mapping since T_1 and T_2 are isomorphic. The mapping of Figure 3 is not a constrained edit mapping since $lca(u_1, u_2) = u_4 \neq lca(u_1, u_3) = u_5$, but $lca(v_1, v_2) = v_4 = lca(v_1, v_3)$, or since $lca(u_1, u_2)$ is not a proper ancestor of u_3, but $lca(v_1, v_2)$ is a proper ancestor of v_3. The mapping of Figure 4 is not a constrained edit mapping since $lca(u_1, u_3) = u_5 = lca(u_2, u_3)$, but $lca(v_1, v_3) = v_5 \neq lca(v_2, v_3) = v_4$, or since $lca(u_2, u_3)$ is a proper ancestor of u_1, but $lca(v_2, v_3)$ is not a proper ancestor of v_1. The *constrained edit distance* (or *structure respecting distance*) between two trees T_1 and T_2 is defined to be $D_c(T_1, T_2) = \min\{\gamma(M_c) | M_c \text{ is a constrained edit mapping between } T_1 \text{ and } T_2\}$.

In this paper, we define a new variant of the edit mapping, called less-constrained edit mapping, by relaxing the restriction of constrained edit mapping such that the edit mapping of Figure 3 is allowable. The *less-constrained edit mapping* M_l is defined to be an edit mapping which satisfies an additional condition that for any triple $(u_1, v_1), (u_2, v_2)$ and (u_3, v_3) in M_l such that none of u_1, u_2 and u_3 is an ancestor of the others, $lca(u_1, u_2) \leq lca(u_1, u_3) = lca(u_2, u_3)$ if and only if $lca(v_1, v_2) \leq lca(v_1, v_3) = lca(v_2, v_3)$. For example, the mapping of Figure 3 is a less-constrained edit mapping since $lca(u_1, u_2) < lca(u_1, u_3) = lca(u_2, u_3)$ and $lca(v_1, v_2) = lca(v_1, v_3) = lca(v_2, v_3)$. However, the mapping of Figure 4 is not a less-constrained edit mapping since $lca(u_1, u_2) < lca(u_1, u_3) = lca(u_2, u_3)$, but $lca(v_1, v_2) = lca(v_1, v_3) > lca(v_2, v_3)$. We define the *less-constrained edit distance* between two trees T_1 and T_2 as $D_l(T_1, T_2) = \min\{\gamma(M_l) | M_l \text{ is a less-constrained edit mapping between } T_1 \text{ and } T_2\}$.

The *editing* (respectively, *constrained editing* and *less-constrained editing*) *problem* is the problem of computing $D_e(T_1, T_2)$ (respectively, $D_c(T_1, T_2)$ and $D_l(T_1, T_2)$) between two trees T_1 and T_2. The editing problem is shown to

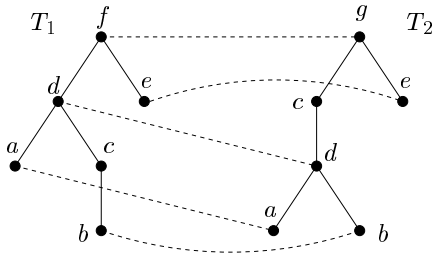


Fig. 1. The dashed lines denote an edit mapping.

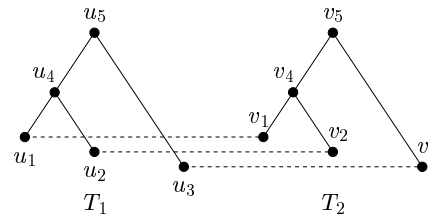


Fig. 2. An edit mapping which is constrained and less-constrained.

be NP-complete [11] and even be MAX SNP-hard [9] for two unordered labeled trees, but is solvable in $O(|T_1| \times |T_2| \times \min\{depth(T_1), leaves(T_1)\} \times \min\{depth(T_2), leaves(T_2)\})$ time for two ordered labeled trees [10]. The constrained editing problem can be solved in $O(|T_1| \times |T_2| \times (degree(T_1) + degree(T_2)) \times \log(degree(T_1) + degree(T_2)))$ time for unordered labeled trees [8] and in $O(|T_1| \times |T_2|)$ time for ordered labeled trees [7]. In this paper, we show that the less-constrained editing problem for unordered labeled trees is NP-complete and even has no absolute approximation algorithm, unless P = NP. The negative result of absolute approximation also implies that it is very unlikely to have a PTAS for the problem. In fact, we can show this problem to be MAX SNP-hard. However, this result does not necessarily imply that it has no absolute approximation algorithm. For ordered labeled trees, we give an algorithm of $O(|T_1| \times |T_2| \times degree^3(T_1) \times degree^3(T_2) \times (degree(T_1) + degree(T_2)))$ time to solve it.

2 NP-Completeness Result

LCEP (Less-Constrained Editing Problem on Unordered Labeled Trees)
Instance: Two unordered labeled tree T_1 and T_2 and a positive integer K.
Question: Is the less-constrained edit distance between T_1 and $T_2 \leq K$?
X3C (Exact Cover by 3-Sets Problem)
Instance: A finite set X with $|X| = 3n$ and a collection $\mathcal{S}$ of 3-element subsets of X with $|\mathcal{S}| = m$.
Question: Does $\mathcal{S}$ contain an exact cover for X, i.e., a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that every element of X occurs in exactly one member of $\mathcal{S}'$?

Note that X3C is a NP-complete problem [3]. Let $X = \{x_1, x_2, \cdots, x_{3n}\}$ and $\mathcal{S} = \{S_1, S_2, \cdots, S_m\}$ be an instance of X3C. Without loss of generality, we assume that $m > n$. Then we transform X and $\mathcal{S}$ into an instance of LCEP, T_1 and T_2, as shown in Figure 5 and 6, respectively. For convenience, we let $S_i = \{t_{i,1}, t_{i,2}, t_{i,3}\}$ for $1 \leq i \leq m$ and call a node with label α an *α-node*.

The root of T_1 is an R-node which has m S-nodes as the children to correspond to $S_1, S_2, \cdots S_m$ of $\mathcal{S}$. Each S-node has three children which are labeled $t_{i,1}, t_{i,2}$ and $t_{i,3}$ respectively if it corresponds to S_i. The root of T_2 is an R-node and has $m + 2n$ children in which $3n$ children are labeled $x_1, x_2, \cdots, x_{3n}$ respec-

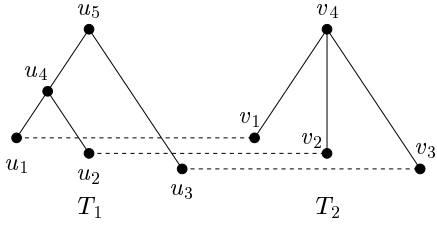


Fig. 3. An edit mapping which is less-constrained, but not constrained.

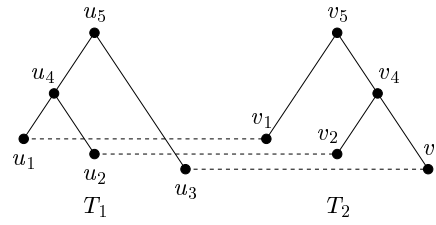


Fig. 4. An edit mapping which is neither constrained nor less-constrained.

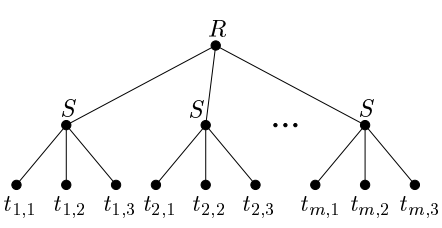


Fig. 5. The unordered labeled tree T_1.

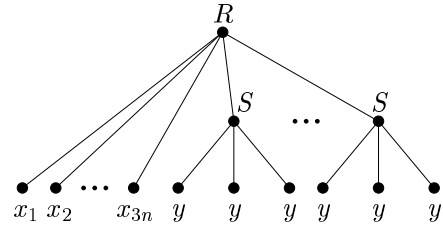


Fig. 6. The unordered labeled tree T_2.

tively and other $m - n$ children are S-nodes. Each S-node has three y-nodes as the children. Clearly, the construction of T_1 and T_2 can be accomplished in polynomial time. For convenience, we assume that the cost of each edit operation is 1, i.e., $\gamma(u \to v) = 1$ if $u \neq v$.

Lemma 1. *Let M be any less-constrained edit mapping from T_1 to T_2. Suppose that there are k, $1 \leq k \leq |T_2|$, uncovered nodes in T_2. Then $\gamma(M) \geq 3m-2n+k$.*

Proof. Let U and V be the sets of the uncovered nodes in T_1 and T_2, respectively. Then $|V| = k$. Since $|T_1| = 4m + 1$, $|T_2| = 4m - n + 1$ and M is a one-to-one mapping, $|M| = |T_2 \setminus V| = 4m-n-k+1$ and $|U| = |T_1|-|M| = n+k$. For each uncovered node in U (V), it corresponds to a delete (insert) operation with cost 1. Hence, U and V totally contribute a cost $n+2k$ to $\gamma(M)$, i.e., $\gamma(M) \geq n+2k$. If $k \geq 3m - 3n$, then $n + 2k \geq 3m - 2n + k$ and hence $\gamma(M) \geq 3m - 2n + k$. Suppose that $k < 3m - 3n$. Then T_2 has at least $(3m - 3n - k)$ y-nodes covered by M. Since T_1 has no y-node, each covered y-node corresponds to a change operation with cost 1. That is, the covered y-nodes in T_2 contribute a cost at least $3m-3n-k$ to $\gamma(M)$. Totally, U, V and the covered y-nodes in $T2$ contribute a cost at least $3m - 2n + k$ to $\gamma(M)$. Hence, $\gamma(M) \geq 3m - 2n + k$. □

Theorem 1. *LCEP is an NP-complete problem.*

Proof. (Sketch) Given an instance X and $\mathcal{S}$ of X3C, we create an instance T_1 and T_2 of LCEP as described in the beginning of this section. First, suppose that $\mathcal{S}$ has an exact cover $\mathcal{S}'$. Then we find a less-constrained edit mapping M from T_1 to T_2 as follows. (1) R-node of T_1 is mapped to R-node of T_2. (2) If S_i is in $\mathcal{S}'$, then we map the children of its corresponding S-node in T_1 to $t_{i,1}$-node, $t_{i,2}$-node and $t_{i,3}$-node in T_2. Note that the S-node is not mapped. (3) If S_i is not in $\mathcal{S}'$, then the subtree of T_1 rooted at the corresponding S-node of S_i is mapped to a subtree of T_2 rooted at an S-node in a way that root is mapped to root and leaf is mapped to leaf. Clearly, the above mapping M is a less-constrained edit mapping and $D_l(T_1, T_2) \leq 3m - 2n$.

Conversely, suppose that there is a less-constrained edit mapping M from T_1 to T_2 with cost $\leq 3m - 2n$. By Lemma 1, all nodes in T_2 are covered by M and hence T_1 contain n uncovered nodes. Then the uncovered nodes of T_1 and

the covered y-nodes of T_2 contribute a total cost $3m - 2n$ to $\gamma(M)$, which leads to $\gamma(M) = 3m - 2n$. Hence, R-node of T_2 is mapped to R-node of T_1, S-node of T_2 is mapped to S-node of T_1, and x_i-node of T_2 is mapped to x_i-node of T_1, $1 \leq i \leq 3n$. Let S' and S'' denote any two S-nodes, which are mapped to each other, in T_1 and T_2, respectively. Then the children of S'' (i.e., 3 y-nodes) are all mapped to the children of S' since the ancestor-descendant relationship is preserved by M. Hence, the uncovered nodes in T_1 are n S-nodes and their children are mapped to x_1-node, x_2-node, $\cdots$, x_{3n}-node in T_2, which implies that these n uncovered S-nodes of T_1 correspond to an exact cover of $\mathcal{S}$. □

3 Negative Result for Absolute Approximation

Recall that Zhang and Jiang [9] showed that the editing problem for two unordered labeled trees is MAX SNP-hard by a reduction from the maximum bounded covering by 3-sets, which is known to be MAX SNP-hard [4]. In fact, their proof can be applied to show that the less-constrained editing problem for two unordered labeled trees is MAX SNP-hard. Note that any MAX SNP-hard problem has no PTAS (Polynomial Time Approximation Scheme), unless P = NP [1], where a problem has a PTAS if for any fixed $\epsilon > 0$, the problem can be approximated within a factor of $1 + \epsilon$ in polynomial time. Hence, it is very unlikely for the less-constrained editing problem to have a PTAS. However, this negative result does not imply that there is no polynomial-time absolute approximation algorithm for the problem. An approximation algorithm is *absolute* if there exists a constant c such that for every instance I of the problem, the *absolute error* $|APP(I) - OPT(I)| \leq c$, where $APP(I)$ and $OPT(I)$ are the approximate and optimal solutions of I, respectively [2]. In this section, we propose a technique of tree composition to show that the less-constrained editing problem for two unordered labeled trees has no polynomial-time absolute approximation algorithm, unless P = NP. For convenience, we assume that the cost of each edit operation is 1, i.e., $\gamma(u \to v) = 1$ if $u \neq v$. For any two forests F_1 and F_2, we use $D_l(F_1, F_2)$ to denote the less-constrained edit distance between them, which is defined analogously to the case of two trees.

Lemma 2. *Let F_1 and F_2 be any two forests of unordered labeled trees. Then we have $||F_1| - |F_2|| \leq D_l(F_1, F_2) \leq |F_1| + |F_2|$.*

Given any k unordered labeled trees $T_1, T_2, \cdots, T_k$, the *composition* of $T_1, T_2, \cdots, T_k$, denoted by $T_1 \circ T_2 \circ \cdots \circ T_k$, is defined to be an unordered labeled tree which is obtained from $T_1, T_2, \cdots, T_k$ by introducing a new root R such that the root of each T_i, $1 \leq i \leq k$, is a child of R (see Figure 7). For convenience, we define $T_1^k = T_1 \circ T_2 \circ \cdots \circ T_k$ iff $T_1 = T_2 = \cdots = T_k$.

Let T_1 and T_2 be any two unordered labeled trees and k be any positive integer. Then we construct two new unordered labeled trees $\mathcal{G}_1 = (T_1 \circ H)^k$ and $\mathcal{G}_2 = (T_2 \circ H)^k$, where H is a complete binary tree with $(2k + 1)(|T_1| + |T_2|)$ y-nodes (see Figure 8 for an example of $\mathcal{G}_1$). In each $\mathcal{G}_i$, $i = 1, 2$, the root has k children and the subtree rooted at each such child is $T_i \circ H$. We denote by $\mathcal{F}_i$

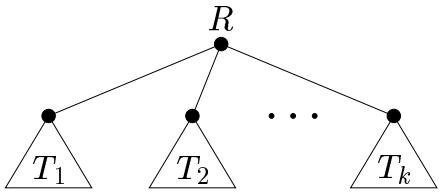


Fig. 7. The composition of $T_1, T_2, \cdots T_k$.

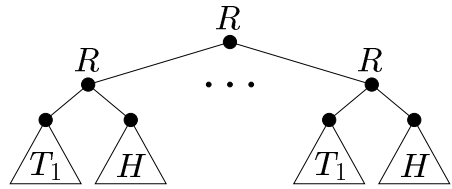


Fig. 8. $\mathcal{G}_1$: its root has k children.

the collection of k subtrees $T_i \circ H$ rooted at the children of the root of $\mathcal{G}_i$. It is not hard to verify the following two lemmas.

Lemma 3. $D_l(\mathcal{G}_1, \mathcal{G}_2) \leq kD_l(T_1, T_2)$.

Lemma 4. *Let M be an arbitrary less-constrained edit mapping with the minimum cost from $\mathcal{G}_1$ to $\mathcal{G}_2$. Then each subtree of $\mathcal{F}_1$ is mapped to exactly one subtree of $\mathcal{F}_2$ by M.*

Lemma 5. $D_l(\mathcal{G}_1, \mathcal{G}_2) \geq kD_l(T_1, T_2)$.

Proof. (Sketch) By Lemma 4, each subtree $A = T_1 \circ H$ of $\mathcal{F}_1$ is mapped to exactly one subtree $B = T_2 \circ H$ of $\mathcal{F}_2$ by an optimal mapping M. Moreover, $root(A)$ and $root(B)$ are mapped each other. We use H_A and H_B to denote H of A and H of B respectively. Suppose that H_A is not covered by M. Then $\gamma(M) \geq |H_A| > kD_l(T_1, T_2)$, a contradiction. Case 1: H_A is mapped to exactly T_2. By Lemma 2, $\gamma(M) \geq |H_A| - |T_2| > kD_l(T_1, T_2)$, a contradiction. Case 2: H_A is mapped to exactly H_B. The distance between A and B must be at least $D_l(T_1, T_2)$. Case 3: H_A is mapped to both T_2 and H_B. Then $root(H_A)$ and T_1 are not covered by M. Hence, there are at least $|T_1| + 1$ uncovered nodes in A and at least $|H_B| + |T_2| - |H_A \setminus \{root(H_A)\}| = |T_2| + 1$ uncovered nodes in B, i.e., the distance between A and B is at least $|T_1| + |T_2| + 2 > D_l(T_1, T_2)$. □

By Lemmas 3 and 5, we have the following theorem.

Theorem 2. $D_l(\mathcal{G}_1, \mathcal{G}_2) = kD_l(T_1, T_2)$ *for any two unordered labeled trees T_1 and T_2.*

Theorem 3. *If there is a polynomial-time absolute approximation algorithm of determining the less-constrained edit distance between two unordered labeled trees, then $P = NP$.*

Proof. Suppose that the problem of determining $D_l(T_1, T_2)$ can be approximated by an absolute approximation algorithm $\mathcal{A}$ with absolute error c in polynomial time $O((|T_1|+|T_2|)^\beta)$ for some constant β. Then we let $k = c+1$ and create $\mathcal{G}_1 =$

$(T_1 \circ H)^k$ and $\mathcal{G}_2 = (T_2 \circ H)^k$ as described previously with each size $O(k^2(|T_1| + |T_2|))$. By applying $\mathcal{A}$ to $\mathcal{G}_1$ and $\mathcal{G}_2$, we can obtain an approximate solution $APP(\mathcal{G}_1, \mathcal{G}_2)$ of $D_l(\mathcal{G}_1, \mathcal{G}_2)$ in time $O(k^{2\beta}(|T_1| + |T_2|)^\beta)$, which is polynomial in $|T_1| + |T_2|$ since k and β are fixed, such that $|APP(\mathcal{G}_1, \mathcal{G}_2) - D_l(\mathcal{G}_1, \mathcal{G}_2)| \leq c$. By Theorem 2, we have $|APP(\mathcal{G}_1, \mathcal{G}_2) - kD_l(T_1, T_2)| \leq c \iff |\frac{APP(\mathcal{G}_1, \mathcal{G}_2)}{k} - D_l(T_1, T_2)| \leq \frac{c}{k}$. Let M denote the corresponding mapping of $APP(\mathcal{G}_1, \mathcal{G}_2)$. Then each subtree $T_1 \circ H$ of $\mathcal{F}_1$ is mapped to exactly one subtree $T_2 \circ H$ of $\mathcal{F}_2$ in M; otherwise, $APP(\mathcal{G}_1, \mathcal{G}_2) \geq |T_i \circ H| > |H| = (2k+1)(|T_1| + |T_2|)$ (where $i = 1$ or $i = 2$), which leads to the absolute error $|APP(\mathcal{G}_1, \mathcal{G}_2) - D_l(\mathcal{G}_1, \mathcal{G}_2)| > (k+1)(|T_1| + |T_2|) > c$, a contradiction. We use $M' \subseteq M$ to denote the mapping from $T_1 \circ H$ to $T_2 \circ H$ with the minimum cost among k such mappings and use H_1 and H_2 to denote H's of $T_1 \circ H$ and $T_2 \circ H$, respectively. Clearly, the cost of $M' \leq \frac{APP(\mathcal{G}_1, \mathcal{G}_2)}{k}$. Note that H_1 must be covered by M'. Case 1: H_1 is mapped to exactly T_2. Then $APP(\mathcal{G}_1, \mathcal{G}_2) \geq |H_1| - |T_2| > 2k(|T_1| + |T_2|)$, which leads to $|APP(\mathcal{G}_1, \mathcal{G}_2) - D_l(\mathcal{G}_1, \mathcal{G}_2)| > k(|T_1| + |T_2|) > c$, a contradiction. Case 2: H_1 is mapped to exactly H_2. In this case, we use the mapping of M' between T_1 and T_2 as the approximate solution of finding the optimal mapping from T_1 to T_2, i.e., $APP(T_1, T_2) \leq \gamma(M') \leq \frac{APP(\mathcal{G}_1, \mathcal{G}_2)}{k}$. Case 3: H_1 is mapped to both T_2 and H_2. Then M' does not cover T_1 and hence $\gamma(M') \geq |T_1| + |T_2| + |H_2| - |H_1| = |T_1| + |T_2|$. In this case, we use $\varnothing$ as the approximate solution of finding the optimal mapping from T_1 to T_2, i.e., $APP(T_1, T_2) = |T_1| + |T_2| \leq \gamma(M') \leq \frac{APP(\mathcal{G}_1, \mathcal{G}_2)}{k}$.

Hence, we find an approximate solution $APP(T_1, T_2)$ of $D_l(T_1, T_2)$ according to $APP(\mathcal{G}_1, \mathcal{G}_2)$ such that $APP(T_1, T_2) \leq \frac{APP(\mathcal{G}_1, \mathcal{G}_2)}{k}$. Then $|APP(T_1, T_2) - D_l(T_1, T_2)| \leq |\frac{APP(\mathcal{G}_1, \mathcal{G}_2)}{k} - D_l(T_1, T_2)| \leq \frac{c}{k} = \frac{c}{c+1} < 1$. Since both $APP(T_1, T_2)$ and $D_l(T_1, T_2)$ are integer-valued, $APP(T_1, T_2) = D_l(T_1, T_2)$, which means that the less-constrained editing problem is solvable by $\mathcal{A}$ in polynomial time and hence we have P = NP by Theorem 1. □

4 A Polynomial-Time Algorithm for Ordered Labeled Trees

We use $D(T_1, T_2)$ instead of $D_l(T_1, T_2)$ if no confusion arises. Given an ordered labeled tree T_i, for convenience, we use the left-to-right postorder numbering to number the nodes of T_i from 1 to $|T_i|$. Following the notation used in [7,8], we use $t_i[k]$ to denote the kth node of T_i, $T_i[k]$ to denote the subtree rooted at $t_i[k]$ and $F_i[k]$ to denote the ordered forest obtained by deleting $t_i[k]$ from $T_i[k]$. Note that the left-to-right order of the trees in $F_i[k]$ is significant. For any two forests $F_1[i]$ and $F_2[j]$, we use $D(F_1[i], F_2[j])$ to denote the less-constrained edit distance between them. Let $\varnothing$ denote the empty tree. Throughout the rest of this section, we let $1 \leq i \leq |T_1|$ and $1 \leq j \leq |T_2|$ and let $t_1[i_1], t_1[i_2], \cdots t_1[i_{n_i}]$ be the children of $t_1[i]$ and $t_2[j_1], t_2[j_2], \cdots t_2[j_{n_j}]$ be the children of $t_2[j]$. Due to the limitation of space, some proofs are omitted in this section.

Lemma 6. (1) $D(\varnothing, \varnothing) = 0$, (2) $D(F_1[i], \varnothing) = \sum_{k=1}^{n_i} D(T_1[i_k], \varnothing)$, $D(T_1[i], \varnothing) = D(F_1[i], \varnothing) + \gamma(t_1[i] \to \lambda)$, (3) $D(\varnothing, F_2[j]) = \sum_{k=1}^{n_j} D(\varnothing, T_2[j_k])$, $D(\varnothing, T_2[j]) = D(\varnothing, F_2[j]) + \gamma(\lambda \to t_2[j])$.

Lemma 7. *Let M be a less-constrained edit mapping from $T_1[i]$ to $T_2[j]$.* (1) *If both $t_1[i]$ and $t_2[j]$ are not covered by M, then $M \cup \{(t_1[i], t_2[j])\}$ is a less-constrained edit mapping.* (2) *If both $t_1[i]$ and $t_2[j]$ are covered by M, then $(t_1[i], t_2[j]) \in M$.*

Lemma 8. $D(T_1[i], T_2[j]) = \min\{D(\varnothing, T_2[j]) + \min_{1 \le k \le n_j}\{D(T_1[i], T_2[j_k]) - D(\varnothing, T_2[j_k])\}, D(T_1[i], \varnothing) + \min_{1 \le k \le n_i}\{D(T_1[i_k], T_2[j]) - D(T_1[i_k], \varnothing)\}, \gamma(t_1[i] \to t_2[j]) + D(F_1[i], F_2[j])\}$.

Proof. (Sketch) Let M be an less-constrained edit mapping with the minimum cost from $T_1[i]$ to $T_2[j]$, i.e., $D(T_1[i], T_2[j]) = \gamma(M)$. By Lemma 7, we can compute $\gamma(M)$ by just considering the following three cases. Case 1: $t_1[i]$ is covered by M, but $t_2[j]$ is not. Case 2: $t_2[j]$ is covered by M, but $t_1[i]$ is not. Case 3: $t_1[i]$ and $t_2[j]$ are both covered by M. □

Suppose that M is any arbitrary less-constrained edit mapping from $F_1[i]$ to $F_2[j]$. In $F_1[i]$ and $F_2[j]$, if no tree is mapped to more than one other tree by M, then we call M a *1-to-1* less-constrained edit mapping; otherwise, we call M a *1-to-many* less-constrained edit mapping. We use $D_1(F_1[i], F_2[j])$ and $D_2(F_1[i], F_2[j])$ to denote the minimum costs of 1-to-1 and 1-to-many less-constrained edit mappings from $F_1[i]$ to $F_2[j]$, respectively. Clearly, we have $D(F_1[i], F_2[j]) = \min\{D_1(F_1[i], F_2[j]), D_2(F_1[i], F_2[j])\}$.

Lemma 9. $D_1(F_1[i], F_2[j]) = \min\{D_1(F_1[i], T_2[j_1] \cdots T_2[j_{n_j-1}]) + D(\varnothing, T_2[j_{n_j}]), D_1(T_1[i_1] \cdots T_1[i_{n_i-1}], F_2[j]) + D(T_1[i_{n_i}], \varnothing), D_1(T_1[i_1] \cdots T_1[i_{n_i-1}], T_2[j_1] \cdots T_2[j_{n_j-1}]) + D(T_1[i_{n_i}], T_2[j_{n_j}])\}$.

Lemma 10. *Let M be a 1-to-many less-constrained edit mapping from $F_1[i]$ to $F_2[j]$. If $T_1[i_k]$ of $F_1[i]$ ($T_2[j_k]$ of $F_2[j]$) is mapped to more than one tree of $F_2[j]$ ($F_1[i]$), then $t_1[i_k]$ ($t_2[j_k]$) is not covered by M.*

Proof. (Sketch) Let $t_1[a_1]$ and $t_1[a_2]$ be any two nodes in $T_1[i_k]$ such that they are mapped to $t_2[b_1]$ of $T_2[j_p]$ and $t_2[b_2]$ of $T_2[j_q]$ respectively. Suppose that $t_1[i_k]$ is mapped to $t_2[b_3]$. Then, due to the ancestor-descendant relationship of M, $t_2[b_3]$ must be a common ancestor of $t_2[b_1]$ and $t_2[b_2]$, which contradicts to the fact that $T_2[j_p]$ and $T_2[j_q]$ are two disjoint trees. □

Lemma 11. *Let M be a 1-to-many less-constrained edit mapping from $F_1[i]$ to $F_2[j]$ and let $T_1[i_k]$ of $F_1[i]$ be mapped to $T_2[j_l] \cdots T_2[j_r]$, $1 \le l < r \le n_j$.* (1) *For any $1 \le p < k$, $T_1[i_p]$ can only be mapped to $T_2[j_1] \cdots T_2[j_{l-1}]$.* (2) *For any $k < q \le n_i$, $T_1[i_q]$ can only be mapped to $T_2[j_{r+1}] \cdots T_2[j_{n_j}]$.*

Proof. (Sketch) (1) Let $t_1[a_1]$ and $t_1[a_2]$ of $T_1[i_k]$ be mapped to $t_2[b_1]$ of $T_2[j_l]$ and $t_2[b_2]$ of $T_2[j_r]$ respectively, and $a_1 < a_2$. Let $t_1[a_3]$ be a node of $T_1[i_p]$ that is mapped to $t_2[b_3]$. Then $a_3 < a_1 < a_2$ and hence $b_3 < b_1 < b_2$. If $t_2[b_3]$ is a node of $T_2[j_l]$, then $lcd(t_2[b_3], t_2[b_1]) < lcd(t_2[b_3], t_2[b_2]) = lcd(t_2[b_1], t_2[b_2])$ and $lcd(t_1[a_3], t_1[a_1]) = lcd(t_1[a_3], t_1[a_2]) > lcd(t_1[a_1], t_1[a_2])$, a contradiction □

Lemma 12. *Let M be a 1-to-many less-constrained edit mapping from $F_1[i]$ to $F_2[j]$ and let $T_2[j_k]$ of $F_2[j]$ be mapped to $T_1[i_l] \cdots T_1[i_r]$, $1 \le l < r \le n_i$. (1) For any $1 \le p < k$, $T_2[j_p]$ can only be mapped to $T_1[i_1] \cdots T_1[i_{l-1}]$. (2) For any $k < q \le n_j$, $T_2[j_q]$ can only be mapped to $T_1[i_{r+1}] \cdots T_1[i_{n_i}]$.*

Lemma 13. $D_2(F_1[i], F_2[j]) =$

$$\min \left\{ \begin{array}{l} \min\limits_{\substack{1 \le k \le n_i \\ 1 \le l < r \le n_j}} \left\{ \begin{array}{l} D(T_1[i_1] \cdots T_1[i_{k-1}], T_2[j_1] \cdots T_2[j_{l-1}]) \\ +D(F_1[i_k], T_2[j_l] \cdots T_2[j_r]) + \gamma(t_1[i_k] \to \lambda) \\ +D(T_1[i_{k+1}] \cdots T_1[i_{n_i}], T_2[j_{r+1}] \cdots T_2[j_{n_j}]) \end{array} \right\}, \\ \min\limits_{\substack{1 \le k \le n_j \\ 1 \le l < r \le n_i}} \left\{ \begin{array}{l} D(T_1[i_1] \cdots T_1[i_{l-1}], T_2[j_1] \cdots T_2[j_{k-1}]) \\ +D(T_1[i_l] \cdots T_1[i_r], F_2[j_k]) + \gamma(\lambda \to t_2[j_k]) \\ +D(T_1[i_{r+1}] \cdots T_1[i_{n_i}], T_2[j_{k+1}] \cdots T_2[j_{n_j}]) \end{array} \right\} \end{array} \right\}.$$

By above lemmas, we design the following algorithm to solve the problem in $O(|T_1| \times |T_2| \times degree^3(T_1) \times degree^3(T_2) \times (degree(T_1) + degree(T_2)))$ time.

1: $D(\varnothing, \varnothing) = 0$, $D_1(\varnothing, \varnothing) = 0$, $D_2(\varnothing, \varnothing) = 0$; **/* Initialization */**
  for $i = 1$ to $|T_1|$ **do**
    $D(F_1[i], \varnothing) = \sum_{k=1}^{n_i} D(T_1[i_k], \varnothing)$; $D(T_1[i], \varnothing) = D(F_1[i], \varnothing) + \gamma(t_1[i] \to \lambda)$;
  for $j = 1$ to $|T_2|$ **do**
    $D(\varnothing, F_2[j]) = \sum_{k=1}^{n_j} D(\varnothing, T_2[i_k])$; $D(\varnothing, T_2[j]) = D(\varnothing, F_2[j]) + \gamma(\lambda \to t_2[j])$;
2: **for** $i = 1$ to $|T_1|$ **do /* The computation of $D(T_1[i], T_2[j])$ */**
    for $j = 1$ to $|T_2|$ **do**
      for $p_1 = 1$ to n_i **do**
        for $p_2 = p_1$ downto 1 **do**
          for $q_1 = 1$ to n_j **do**
            for $q_2 = q_1$ downto 1 **do**
              /* Let $F_1 = T_1[i_{p_2}] \cdots T_1[i_{p_1}]$ and $F_2 = T_2[i_{q_2}] \cdots T_2[i_{q_1}]$. */
              Compute $D_1(F_1, F_2)$ by Lemma 9 and $D_2(F_1, F_2)$ by Lemma 13;
              $D(F_1, F_2) = \min\{D_1(F_1, F_2), D_2(F_1, F_2)\}$;
      end for
      Compute $D(T_1[i], T_2[j])$ by Lemma 8;
    end for
  end for
3: Output $D(T_1[|T_1|], T_2[|T_2|])$.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, Proof verification and the hardness of approximation problems, *Journal of the Association for Computing Machinery*, **45** (1998) 501–555.
2. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamelai, and M. Protasi, *Complexity and Approximation—Combinatorial Optimization Problems and Their Approximability Properties*, Springer Verlag, Berlin, 1999.
3. M.R. Garey and D.S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, San Francisco, Freeman, 1979.
4. V. Kann, Maximum bounded 3-dimensional matching is MAX SNP-complete, *Information Processing Letters*, **37** (1991) 27–35.
5. T. Richter, A new measure of the distance between ordered trees and its applications, Technical Report 85166-CS, Department of Computer Science, University of Bonn, 1997.
6. E. Tanaka and K. Tanaka, The tree-to-tree editing problem, *International Journal of Pattern Recognition and Artificial Intelligence*, **2** (1988) 221–240.
7. K. Zhang, Algorithms for the constrained editing distance between ordered labeled trees and related problems, *Pattern Recognition*, **28** (1995) 463–474.
8. K. Zhang, A constrained edit distance between unordered labeled trees, *Algorithmica*, **15** (1996) 205–222.
9. K. Zhang and T. Jiang, Some MAX SNP-hard results concerning unordered labeled trees, *Information Processing Letters*, **49** (1994) 249–254.
10. K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM Journal on Computing*, **18** (1989) 1245–1262.
11. K. Zhang, R. Statman, and D. Shasha, On the editing distance between unordered labeled trees, *Information Processing Letters*, **42** (1992) 133–139.

A Highly Efficient Algorithm to Determine Bicritical Graphs*

Dingjun Lou and Ning Zhong

Department of Computer Science
Zhongshan University
Guangzhou 510275
People's Republic of China

Abstract. In this paper, we show a necessary and sufficient condition which characterizes all bicritical graphs. Using this necessary and sufficient condition we develop a highly efficient algorithm to determine whether a graph is bicritical, of which the time complexity is bounded by $O(|V||E|)$, and this is the best result that has ever been known.

1 Introduction and Terminology

All graphs considered in this paper are finite, undirected, connected and simple. In general, we follow the terminology of [1]. A graph G is said to be bicritical if $G-u-v$ has a perfect matching for each pair of vertices u and v in G such that $u \neq v$. Plummer[10] introduced the concept of n-extendable graphs. Let G be a graph on v vertices and n be an integer such that $0 \leq n \leq (v-2)/2$. Then G is n-extendable if every matching of size n in G is contained in a perfect matching of G.

Bicritical graphs play a central role in the decomposition theory of graphs in terms of their maximum matchings. In [10], it is proved that every n-extendable graph is also $(n-1)$-extendable and also that any 2-extendable graph is either 1-extendable bipartite or bicritical. Thus for non-bipartite graphs, if we let G_k denote the class of those which are k-extendable and G_b those which are bicritical, we have a nested sequence of graph classes:$G_1 \supset G_b \supset G_2 \supset G_3 \ldots$. Moreover, it is known that each of the containments is proper. In [7], Lovsz and Plummer point out that the structure of bicritical graphs is far from completely understood. We show a necessary and sufficient condition to characterize all bicritical graphs in this paper.

The concept of bicritical graphs is also generalized to n-critical graphs and there has been some research on this topic (see [2]-[6]). In [12], Yu gives a Tutte style necessary and sufficient condition for n-critical graphs. However, it does not help to design an efficient algorithm to determine the n-critical graphs. Using our necessary and sufficient condition, we can design a highly efficient algorithm to determine all bicritical graphs.

* The work of this paper was supported by the National Science Foundation of China.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 349–356, 2001.
© Springer-Verlag Berlin Heidelberg 2001

2 A Necessary and Sufficient Condition for Bicritical Graphs

In this section, we give a necessary and sufficient condition for bicritical graphs.

Theorem 1. *Let G be a graph with a perfect matching M_0, then the following propositions are equivalent:*

1. G is bicritical;
2. For any perfect matching M and any two different vertices x and y in G, there is an M-alternating path between x and y, which starts and ends with edges in $E(G) \setminus M$
3. For every pair of vertices x and y in G, there is an M_0-alternating path between x and y, which starts and ends with edges in $E(G) \setminus M_0$.

Proof. We show that (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1).

1. (1) $\Rightarrow$ (2): Suppose that G is bicritical, then for any perfect matching M in G and any two vertices $u, v \in V(G)$, $G - \{u, v\}$ has a perfect matching. For any pair of vertices $x, y \in V(G)$, we discuss in two cases:

   Case 1. $xy \notin M$. Suppose that $xu, yv \in M$. $G - \{u, v\}$ has a perfect matching. Let $M_1 = M - \{xu, yv\}$ be a matching in $G - \{u, v\}$, then both x and y are M_1-unsaturated. So, there must be an M_1-augmenting path between x and y, otherwise, according to Berge's theorem, M_1 is the maximum matching in $G - \{u, v\}$, which contradicts the fact that G is bicritical.

   Case 2. $xy \in M$. For $yw \in E(G) \setminus M$, since G is 1-extendable, yw is contained in a perfect matching M'. Let $G_1 = G[M \Delta M']$, where $M \Delta M'$ denotes the symmetric difference of M and M'. Then the component of G_1 that contains xy is an even cycle with edges alternatively in M and M', and we have an M-augmenting path between x and y as desired. (see Figure 2).
2. (2) $\Rightarrow$ (3) follows trivially since (3) is a special case of (2).
3. (3) $\Rightarrow$ (1) For any two vertices u, v in $V(G)$, if $uv \in M_0$, then $G - \{u, v\}$ has a perfect matching $M_0 \setminus \{uv\}$. Suppose $uv \notin M_0$, and $xu,\ yv \in M_0$. By (3), there is an M_0-alternating path P between vertices x and y, which starts and ends with edges in $E(G) \setminus M_0$, and P is an $M_0 \setminus \{xu, yv\}$-augmenting path in $G - \{u, v\}$. By exchanging edges in M_0 and edges not in M_0 along P, we have a perfect matching in $G - \{u, v\}$.Thus, G is bicritical.

We call such an M-alternating path P between two M-saturated vertices x and y, which starts and ends with edges in $E(G) \setminus M$ a pseudo-M-alternating path.

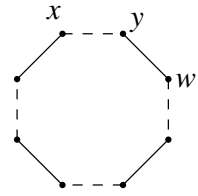


Fig. 1. $G[M \Delta M']$, M broken lines, M' solid lines.

3 Description of the Algorithm

First, we can use an $O(\sqrt{n}\,|E|)$ algorithm at hand to find a perfect matching M_0 in a given graph G (see [8]), if such M_0 cannot be found, then G is not bicritical, and we can make our conclusion at once. Otherwise, for any two vertices, say, x and y, we try to find an M_0-alternating path between them in G, which starts and ends with edges in $E(G) \setminus M_0$.This should be done for all pairs of vertices, and if there exists such a path between every pair of vertices in G, then according to Theorem 1, G is bicritical, otherwise, G is not bicritical. To determine bicritical graphs, we develop an algorithm, MAPS(x, exist) in the main recognition program to find all the M_0-alternating paths which start and end with edges in $E(G) \setminus M_0$ in G between a given vertex, say x and every other vertex in G in $O(|E|)$ time.

The main recognition program to determine bicritical graphs

1. exist:=true; // $O(1)$ time
2. Construct an adjacency list for G; // $O(|E|)$ time
3. Try to find a perfect matching M_0 in G; //$O(\sqrt{n}\,|E|)$ time
4. If such M_0 does not exist then exit; (G is not bicritical)
5. **else for** each of $|V|-1$ vertices in G, say x **do**
   begin
6. MAPS(x, exist); // pseudo-M_0-alternating path search procedure
7. **If** exist=false then exit (G is not bicritical') // $O(1)$ time
   end
8. Exit (G is bicritical) // $O(1)$ time

Our algorithm MAPS procedure is derived from the idea inspired by Edmond, there are some differences though, mainly in the handling of blossoms and in the way we extend an M_0-alternating tree. We use an adjacency list to store G. Suppose that all vertices have indices in sequence, and for all vertices, there exists a table in the storage, which is of the form below.

Vertex (i)	1	2	3	...	n
m					
f					
b					
p					

In this table, for any given vertex i, $m(i)$ denotes the other vertex adjacent to i with an edge in M_0. $f(i)$ represents the father of i, that is, in the construction of the alternating tree, the predecessor of i, and $b(i)$ is the root of the blossom that contains i. Finally, if there is a pseudo-M_0-alternating path between x and i when constructing an M_0-alternating tree rooted with x, then $p(i) = 1$, otherwise $p(i) = 0$. Initially, let $f(i) := 0$; $b(i) := i$; $p(i) = 0$ for all i.

Construct the adjacency list and the table for G, which needs $O(|E|)$ time.

An alternating tree with root x with respect to a matching M is a subgraph of G, $Tree = (V, T)$, which is connected and without a cycle, such that:

1. $x \in v$ is the root of the tree, which is M-unsaturated;
2. for each $i \in V$, the unique path $L_T(i)$ of the tree between x and i is an alternating path.

The alternating tree of Figure 3 illustrates this definition.

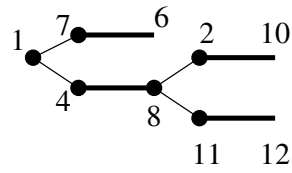


Fig. 2. An alternating tree with root $x = 1$. Heavy edges represent matching edges

We shall say that the vertices of G which belong to V are labelled. The vertices of $V(G) - V$ are called unlabelled. Let $i \in V$ be a labelled vertex.

- If the path $L_T(i)$ has an even number of edges, then we say that the vertex i is even. In particular, x, the root of the tree is even.
- If the path $L_T(i)$ has an odd number of edges, then the vertex i is said to be odd. In an alternating tree, the odd vertices must all be of degree 2: this is precisely because there is always exactly one thin and one heavy edge incident to such a vertex. What's more, it's crucial to point out that if vertex i is labelled odd, then there must exist a pseudo-M_0-alternating path between x and i.
- If one walks through an alternating tree from a vertex i to the root x, then the vertices which one meets are alternatively even and odd.

Now, we are ready to introduce our pseudo-M_0-alternating path search procedure (MAPS) between x and every other vertex in G. We use breadth-first-search method (BFS) to handle every vertex in G. In each MAPS procedure, we need an even vertices queue V_0 to store even vertices. Denote by $V_0 \subset V(G)$ the subset of vertices labelled even. The construction of the alternating tree will proceed step by step, starting from

$$V = V_0 = \{x\}, T = \emptyset$$

At each stage, Tree $= (V, T)$ induces an alternating tree on G. The MAPS procedure is as follows:

The pseudo-M_0-alternating path search procedure (MAPS) between x and every other vertex in G.

1. Set the even vertices queue V_0 empty;
2. Store x in even vertices queue V_0,label x even and set $p(x) := 1$;

3. Remove the first vertex i from V_0;
4. Use BFS to handle i, that is, for every edge $e = (i, j)$ which is incident to i and j is not labelled odd, two cases are possible:
5. **Case** 1: j is unlabelled;
   Label j odd;
   Set $p(j) := 1$;
   Get $m(j)$ from the table;
   If $m(j)$ is unlabelled then
   begin
   Add $m(j)$ to the even vertices queue V_0;
   Write $f(j) := i$, $f(m(j)) := j$ to the table;
   Label $m(j)$even;
   end
   end of case 1;
   // $T \leftarrow T + \{e\} + \{jm(j)\}$
   // $V \leftarrow V + \{j\} + \{m(j)\}$
   //$V_0 \leftarrow V_0 + \{m(j)\}$
6. **Case** 2: j is also labelled even ($j \in V_0$)
   $i' := i$; $j' := j$;
7. **While** $i' \neq b(i')$ or $j' \neq b(j')$ **do**
   begin
   $i' := b(i')$;
   $j' := b(j')$;
   end
   If $i' = j'$ then we do nothing **else**
   begin
   $k := i', l := j'$;
8. **While** (neither i' nor j' has been marked) and ($i' \neq j'$) **do**
   begin
   Marked i' and j'
   $i' := b(f(i'))$;
   $j' := b(f(j'))$;
   end
   If j' has been marked then $r := j'$
   else $r = i'$
   end

Note 1. r, which has at least degree 3 (unless $r = x$), must be labelled even. The subpath L_i and L_j of $L_T(i)$ and $L_T(j)$, on the one hand between r and i, and on the other hand between r and j, are even and $\mu = L_i + L_j + \{e\}$ is certainly of odd cardinality. The matching $M\mu = M_0 \cap \mu$ is a maximum matching of μ and r, the only $M\mu$-unsaturated vertex of the cycle, is linked with x by an even alternating path $L_T(r)$. Such a cycle μ is called a blossom of M_0, and the vertex r is called the root of the blossom. When a blossom has been found, each vertex in it must have a pseudo-M_0-alternating path that links it to the root x. Now we use a supplementary queue V_1 to store the vertices of μ that are labelled odd and notice that we need not make any changes to those vertices that have already been included in some other blossoms. This can be done as follows.

9. **While** $k \neq r$ do
   begin
   If k is labelled odd then
   begin
   Put k into V_1;
   Re-label k even
   end
   else $p(k) := 1$;
   $b(k) := r$
   Write $b(k)$ to the table;
   $k := f(k)$
   end
10. **While** $l \neq r$ **do**
   begin
   If l is labelled odd then
   begin
   Put l into V_1;
   Re-label l even
   end
   else $p(l) := 1$
   $b(l) := r$;
   Write $b(l)$ to the table
   $l := f(l)$
   end
11. Add V_1 to the tail of V_0;
   end of case 2;
12. **If** V_0 is not empty then go to 4
13. **for** every vertex i of G **do**
   if $p(i) = 0$ **then** exit(exist:=false);
14. exit(exist:=true).

4 The Correctness and the Time Complexity of the Program

Lemma 1. *If there exist pseudo-M_0-alternating paths between x and every other vertex y in G, then MAPS can find them.*

Proof. If y is labelled odd then the lemma trivially follows. If y is labelled even then only when it is contained in a blossom, there is a pseudo-M_0-alternating path between x and y. The last case is that the M_0-alternating tree rooted with x fails without reaching y. In this case, we prove that there is no pseudo-M_0-alternating path between x and y in G as follows:

Let u, v be the vertices that are adjacent to x and y respectively, with matching edges. If there exists an M_0-alternating path between x and y in $G - \{u, v\}$, then it must be an M_0-augmenting path. When the alternating tree with

root x fails to reach y, the matching is maximum (with only x and y M_0-unsaturated). Thus, according to Berge's theorem, there is no M_0-augmenting path in $G - \{u, v\}$. So the lemma follows.

Theorem 2. *The program that determines whether any given simple graph G is bicritical is correct.*

Proof. For every pair of vertices x and y in G, the program is to check whether there exits a pseudo-M_0-alternating path between x and y, which starts and ends with edges in $E(G) \setminus M_0$. Thus, the theorem is a direct result from Theorem 1 and Lemma 1

Theorem 3. *In the worst case, the time complexity of the program is bounded by $O(|V|\,|E|)$.*

Proof. In the following proof, denote $|V|$ by N. First, let's look at the main recognition program to determine bicritical graphs: obviously, program sections 5–7 are a loop that checks each vertex in G, in the worst case, we must construct $N-1$ alternating trees. The problem is therefore that of determining the number of elementary operations (additions, comparisons, and access to storeage) needed to construct an M_0-alternating tree, that is, the time complexity of the procedure MAPS.

The structure of the graph G is supposed to be described by an adjacency list giving, for each vertex i, those vertices which are adjacent to i. Each edge $e = (i, j)$ appears thus twice in the list.

In the procedure MAPS, program sections 1–3 need $O(1)$ time. Each edge is examined at most twice in our BFS in accordance to their appearance in the adjacency list. So the number of operations that work on the adjacency list is proportional to $|E|$.

When a blossom μ_i has been detected, the number of operations necessary to change the labels and the updates $b(i)$ of the vertices of the blossom is proportional to $|\mu_i|$. Notice that the number of blossoms which might appear in the course of the construction of an alternating tree is at most $\lfloor N/2 \rfloor$ (the integer part of $N/2$). Since every vertex is handled at most once in a blossom, the total sum of $|\mu_i|$, i from 1 to ν, is at most $N + N/2$ ($N/2$ is the maximal number of the roots of all blossoms). Thus, the number of all operations of this type is proportional to N.

For those edges that won't form blossoms when they are added to the alternating tree, the time complexity over all is $O(k)$, k is the number of such edges, and for those edges that will form blossoms when they are added to the alternating tree, the time complexity over all is $O(N + N/2)$, thus in the procedure MAPS from program sections 4–12, we need at most $O(k + N + N/2)$ time, that is, $O(|E|)$ time to construct an M_0-alternating tree, Section 13 needs $O(N)$ time and section 14 needs $O(1)$ time, so our MAPS needs $O(|E|)$ time, and for all N alternating trees, we need $O(N\,|E|)$.Thus the theorem follows.

5 Remarks

If we use the algorithm derived from the definition of bicritical graphs, that is, choose every pair of vertices x and y of G, delete x and y together with edges incident to them, and then check whether $G-\{u,v\}$ has a perfect matching, then the time complexity of such an algorithm will come out as $O\left(N^{5/2}\,|E|\right)$, which is obviously inferior to the algorithm we get above, with the space complexity almost the same. So, our algorithm has higher efficiency.

References

1. J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, Macmillan Press, London (1976).
2. Guizhen Liu and Qinglin Yu, Toughness and perfect matchings in graphs, Ars Combinatorics 48 (1998), 129–134.
3. Guizhen Liu and Qinglin Yu, On n-edge-deletable and n-critical graphs, Bulletin of the Institute of Combinatorics and Its Applications, (to appear).
4. Dingjun Lou, On matchability of graphs, Australasian Journal of Combinatorics 21 (2000), 201–210.
5. Dingjun Lou and Qinglin Yu, Sufficient condition for n-critical graphs, 1998, (submitted).
6. L. Lovsz and M. D. Plummer, On a family of planar bicritical graphs, Proc. London Math. Soc. (3) 30 (1975), 160–176.
7. L. Lovsz and M. D. Plummer, Matching Theory, Elsevier, North-Holland, Amsterdam, 1986.
8. S. Micali and V. V. Vazirani, An $O\left(\sqrt{n}\,|E|\right)$ algorithm for finding maximum matchings in general graphs, 21st Annual Symposium on Foundations of Computer Science, Syracuse, NY, pp.17–27 (NY,USA,IEEE 1980).
9. Michel Gondran and Michel Minoux, Graphs and Algorithms, John Wiley & Sons Ltd. (1984).
10. M. D. Plummer, On n-extendable graphs, Discrete Math. 31(1980), 201–210.
11. M. D. Plummer, Extending Matchings in Graphs: An Update, Congressus Numerantium 116 (1996), pp.3–32
12. Qinglin Yu, Factors and factor extensions, Doctoral dissertation, Simon Fraser University (1991).

Layered Drawings of Graphs with Crossing Constraints*

Irene Finocchi

Department of Computer Science
University of Rome "La Sapienza"
finocchi@dsi.uniroma1.it
http://www.dsi.uniroma1.it/~finocchi

Abstract. We study the problem of producing hierarchical drawings of layered graphs when some pairs of edges are not allowed to cross. We show that deciding on the existence of a drawing satisfying at least k constraints from a given set of non-crossing constraints is NP-complete even if the graph is 2-layered and even when the permutation of the vertices on one side of the bipartition is fixed. We also propose simple constant-ratio approximation algorithms for the optimization version of the problem and we discuss how to extend the well-known hierarchical approach for creating layered drawings of directed graphs with the capability of minimizing the number of edge crossings while maximizing the number of satisfied non-crossing constraints.

1 Introduction

The problem of embedding graphs in the plane when only some pairs of edges are allowed to cross, known in literature as *realizability*, has been formally introduced in [12]. It finds application, e.g., in VLSI layout, where the crossings between certain pairs of edges must be avoided due to the physical realization of connectors, and in constrained graph layout, where users can specify requirements on the visualization that should be fulfilled by the drawing algorithm. In these settings, it is fundamental to decide if an embedding of a graph satisfying a given set of non-crossing constraints exists, and, if the answer is positive, to find it. Realizability has been proved to be NP-hard in [10], but surprisingly it is not known to belong to the class NP, turning out to be a very interesting problem from a theoretical perspective. Its theoretical relevance is mostly due to the connection with the classical themes of planarity and crossing numbers and with the recognition of string graphs, i.e., intersection graphs of curves in the plane. The relation between realizability and string graphs is addressed in [10,11]; here we limit to point out that no finite algorithm for the recognition of string graphs is known and that, given a graph G, it is possible to build an instance of realizability which can be satisfied if and only if G is a string graph [11]. It is also worth

* Work supported in part by the project "Algorithms for Large Data Sets: Science and Engineering" of the Italian Ministry of University and of Scientific and Technological Research (MURST 40%).

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 357–368, 2001.
© Springer-Verlag Berlin Heidelberg 2001

observing that realizability is a generalization of the well-known planarity problem, which requires to decide if a graph is planar and clearly coincides with the special case where no pair of edges is allowed to cross. We recall that planarity can be checked in linear time [9].

This paper is concerned with the realizability problem in the special case of hierarchical drawings [3], i.e., drawings where vertices are constrained to lie on a set of parallel lines and edges are represented as polygonal chains. The hierarchical drawing convention is widely used to represent procedure call dependencies, class hierarchies, is-a relationships, and several economic and social structures [3,5,14]. In the remainder of the paper we will refer to the vertex set on each line as *layer* or *level*, and sometimes to the drawing itself as *layered drawing*. Without loss of generality we also assume to deal with *proper* layered graphs, i.e., layered graphs whose edges span only consecutive levels: it is always possible to reduce to this case by properly adding dummy vertices to split edges with end-points in non-consecutive layers. Under this hypothesis each edge in a hierarchical drawing is simply represented by a straight-line. The main question that we address is the following: *Given a proper layered graph G, a set of edge pairs C, and an integer $k > 0$, does a hierarchical drawing of G exist s.t. at least k edge pairs in C do not cross?* We refer to this problem as *hierarchical realizability* (HR) and to the elements of C as *non-crossing constraints*. We study the computational complexity of some variants of hierarchical realizability and we describe approximation strategies for the optimization versions of these problems, that require to find a maximum realizable subset of C. We also show how algorithms for the variants that we consider can be used as subroutines to boost existing graph drawing algorithms to support non-crossing constraints. Our results can be easily generalized to constraints concerning the relative positions of vertices within each layer, though for simplicity we will not consider them in this paper.

In more detail, the presentation of our results is organized as follows. In Section 2 we prove that deciding on the realizability in case of hierarchical drawings is not easier than deciding on the realizability for generic drawings: we show that HR, which clearly belongs to NP, remains NP-hard even if G is 2-layered and $k = |C|$. It is interesting to observe that the strictly related problem of deciding if a layered graph has a hierarchical drawing *without* edge crossings can be solved in linear time [8]. From an optimization point of view, it is trivial to approximate a maximum cardinality subset of realizable constraints with expected ratio 2, yet we can exhibit instances for which this bound is tight. In addition to the general case, in Section 3 we consider a one-sided realizability problem where G is 2-layered and the vertex ordering of a layer is fixed; we show this problem to be NP-hard for a generic k, approximable with constant ratio 2, and polynomial if $k = |C|$. Based on the one-sided formulation, in Section 4 we finally address the constrained edge crossing minimization problem in layered drawings, i.e., the problem of drawing hierarchically a graph with the minimum number of edge crossing while respecting a set of non-crossing constraints. We consider the well-known Sugiyama's approach for producing layered drawings of directed

graphs [14] and we extend it in order to deal with non-crossing constraints. In particular, we provide a mechanism for identifying a subset $C' \subseteq C$ of realizable constraints, and then we show how to minimize edge crossings while satisfying all the constraints in C'.

2 Hierarchical Realizability

In this section we prove that deciding about the hierarchical realizability of a proper layered graph under a given set of non-crossing constraints is NP-complete. From an optimization point of view, we show that a maximum cardinality subset of realizable constraints can be easily approximated with expected ratio 2 and we exhibit instances for which this bound is tight.

Lemma 1. *HR* $\in NP$.

Proof. Since edge crossings in layered drawings depend only on the permutation of the vertices within each layer, a non-deterministic algorithm has simply to guess an ordering of the vertices of each layer and to check in polynomial time the realization of at least k constraints.

In the following we show the NP-hardness of HR on two-layered (i.e., bipartite) graphs. For brevity, we call bipartite straight-line drawing a hierarchical drawing with only two layers. Let $B = (V_0, V_1, E)$ be a bipartite graph with $n = n_0 + n_1$ vertices and m edges. Without loss of generality, we assume that for any $(a, b) \in E$ $a \in V_0$ and $b \in V_1$. We also denote by π_0 and π_1 two permutations of the vertices in V_0 and V_1, respectively. Let $C \subseteq \binom{E}{2}$ be a set of pairs of edges. The hierarchical realizability problem on bipartite graphs consists of asking for a set $C' \subseteq C$, $|C'| \geq k$, and for a bipartite straight-line drawing of B such that for any $(a, b; c, d) \in C'$ edges (a, b) and (c, d) do not cross in the drawing. More formally, the problem can be stated as follows:

Bipartite realizability (in short, BR): Let $B = (V_0, V_1, E)$ be a bipartite graph, let C be a set of non-crossing constraints, and let k be a positive integer. Do two permutations π_0 and π_1 exist such that the bipartite straight-line drawing of B induced by π_0 and π_1 satisfies at least k constraints?

In the special case where the permutation of the vertices in a layer (e.g., V_0) is fixed, we have the *one-sided bipartite realizability* problem (in short, OBR), whose complexity and approximability are discussed in Section 3.

Before proving the NP-completeness of BR, we state it in a slightly different form which is more convenient for our purposes. Indeed, it is easy to see that asking for two permutations satisfying a set of non-crossing constraints C' is equivalent to asking for two bijective functions π_0 and π_1 such that:

$$\forall (a, x; b, y) \in C' \quad (\pi_0(a) - \pi_0(b)) \cdot (\pi_1(x) - \pi_1(y)) \geq 0 \tag{1}$$

Based on this observation, we prove the NP-hardness of BR by means of a polynomial time reduction from a total ordering problem arising in the computational biology field, called *betweenness*, which has been shown to be NP-complete in [13] and is defined as follows:

Betweenness: Given a finite set S of real variables and a set of triples $T \subseteq S^3$, does an ordering of the variables exist such that for each $(a, b, c) \in T$ variable b is positioned between variables a and c in the ordering?

We refer to the triples in T as *ordering constraints*. A solution for betweenness is a bijective function $f : S \to \{1..|S|\}$ such that:

$$\forall (a, b, c) \in T \quad (f(a) - f(b)) \cdot (f(b) - f(c)) \geq 0 \tag{2}$$

We remark that, in spite of the apparent similarity of Equation 1 and Equation 2, the former involves four distinct vertices and two different functions, while the latter only three variables and a single function. A one-to-one correspondence is therefore not straightforward.

Lemma 2. *Bipartite realizability is NP-hard.*

Proof. Moving from an instance of betweenness, we build an instance of BR as follows. The bipartite graph $B = (V_0, V_1, E)$ is such that: (a) $V_0 = S$; (b) V_1 contains a distinct vertex x' for each item $x \in S$, i.e., $V_1 = \{x' : x \in S\}$; (c) $E = \{(x, x') : x \in S\} \cup \{(a, b'), (b, c') : (a, b, c) \in T\}$. The set of non-crossing constraints C is the union of two disjoint sets C_1 and C_2, where $C_1 = \{(x, x'; y, y') : x, y \in S,\ x \neq y\}$ and $C_2 = \{(a, b'; b, c') : (a, b, c) \in T\}$. Note $|C_1| = |S|(|S| - 1)$ and $|C_2| = |T|$. $k = |C_1| + |C_2|$, i.e., the problem is to decide if *all* the non-crossing constraints can be realized.

It is easy to verify that the reduction above requires $O(|T| + |S|^2)$ time. To prove that k constraints in C can be realized iff the instance of betweenness admits a solution we exploit the one-to-one correspondence between ordering constraints in T and crossing constraints in C_2 and the fact that realizing all the constraints in C_1 forces $\pi_0(a) = \pi_1(a')$ for any pair of corresponding vertices $a \in V_0$ and $a' \in V_1$.

Let us first suppose that two permutations π_0 and π_1 exist able to satisfy all the constraints in C. Then $f = \pi_0$ represents a solution for betweenness. Indeed, let $(a, b, c,) \in T$ be an ordering constraint in the instance of betweenness. Let us consider the constraints $(b, b'; c, c') \in C_1$ and $(a, b'; b, c') \in C_2$: these constraints necessarily exist by construction. Since π_0 and π_1 satisfy all the non-crossing constraints, by Equation 1 we know that: $(\pi_0(a) - \pi_0(b)) \cdot (\pi_1(b') - \pi_1(c')) \geq 0$ and $(\pi_0(b) - \pi_0(c)) \cdot (\pi_1(b') - \pi_1(c')) \geq 0$ By multiplying the left members of these inequalities, we obtain $(\pi_0(a) - \pi_0(b)) \cdot (\pi_0(b) - \pi_0(c)) \geq 0$, proving that the ordering constraint $(a, b, c) \in T$ is satisfied by $f = \pi_0$.

Conversely, let us assume that the instance of betweenness admits a solution f and let us define $\pi_0 = f$ and $\forall a' \in V_1\ \pi_1(a') = f(a)$. Note that $\forall a \in S$ $\pi_0(a) = \pi_1(a')$. Any constraint $(a, a'; b, b') \in C_1$ is trivially satisfied by π_0 and π_1, since $\pi_0(a) - \pi_0(b) = \pi_1(a') - \pi_1(b')$ and therefore their product is ≥ 0. Let $(a, b'; b, c')$ be any constraint in C_2. As above, $(\pi_0(b) - \pi_0(c)) \cdot (\pi_1(b') - \pi_1(c')) \geq 0$. Moreover, since $\pi_0 = f$ satisfies constraint $(a, b, c) \in T$, by Equation 2 $(\pi_0(a) - \pi_0(b)) \cdot (\pi_0(b) - \pi_0(c)) \geq 0$. By multiplying the left members of the two inequalities we finally get $(\pi_0(a) - \pi_0(b)) \cdot (\pi_1(b') - \pi_1(c')) \geq 0$, i.e., constraint $(a, b'; b, c')$ is satisfied by π_0 and π_1, as well.

The NP-completeness of HR immediately follows from Lemma 1 and Lemma 2:

Theorem 1. *Hierarchical realizability is NP-complete.*

We now consider the problem of approximating a maximum set of realizable constraints. If we choose, for each layer of G, a random ordering of the vertices in that layer, we approximate a maximum set of realizable constraints (chosen from C) with expected ratio 2.

If $c = |C|$ and $c_i = (a, x; b, y)$, $1 \leq i \leq c$, is a non-crossing constraint involving vertices from any two consecutive layers L_j and L_{j+1}, four orderings among $a, b \in L_j$ and $x, y \in L_{j+1}$ are possible, yet only two of them realize c_i. Hence, the probability p_i of having constraint c_i satisfied is equal to $\frac{1}{2}$. Let X_i be a random variable equal to 1 if constraint c_i is realized, 0 otherwise. The quantity $\sum_{i=1}^{m} X_i$ represents the number of realized constraints and the expected value for it is then equal to $E[\sum_{i=1}^{c} X_i] = \sum_{i=1}^{c} E[X_i] = \sum_{i=1}^{c} p_i = \frac{c}{2}$.

We now show that the ratio above is tight, i.e., no algorithm can do better than trying to satisfy half of the constraints in C, unless being able to recognize non-realizable instances of HR. The following lemma is useful at this aim:

Lemma 3. *The number of non-crossing pairs of non-incident edges in any bipartite straight-line drawing of $K_{n,n}$ is equal to $\left(\frac{n(n-1)}{2}\right)^2$.*

Proof. Any bipartite straight-line drawing of $K_{n,n}$ has the same number x of non-crossing pairs of non-incident edges. Then:

$$x = \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\sum_{k<i} \sum_{h<j} 1 + \sum_{k>i} \sum_{h>j} 1 \right) = \left(\tfrac{n(n-1)}{2}\right)^2$$

Let $G = K_{n,n}$ and let C contain a constraint for any pair of non-incident edges. It is easy to see that $c = |C| = \frac{n^2(n-1)^2}{2}$ and therefore, due to Lemma 3, the number of realized constraints is always equal to $\frac{c}{2}$ independently of the permutation of the two layers.

3 One-Sided Bipartite Realizability

In this section we focus on the one-sided bipartite realizability problem, that turns out to be very useful in the design of algorithms for solving HR, as we will show in Section 4. Unfortunately, we can prove that it is NP-complete, yet easier than BR, being polynomially solvable if we are interested in the existence of a solution satisfying *all* the constraints in C (i.e., for $k = |C|$).

NP-completeness. For simplicity we prove the NP-hardness of one-sided bipartite realizability in two steps, making use of a simple generalization of this problem which takes into account also *crossing constraints* in addition to the non-crossing ones. We refer to the generalized problem as *one-sided bipartite realizability with mixed constraints* (in short, MBR). MBR is stated exactly as OBR, except for taking into account a set of crossing constraints $I \subseteq \binom{E}{2}$ with

the following meaning: for any $(a, b; c, d) \in I$, edges (a, b) and (c, d) should cross in the bipartite drawing of graph B.

With arguments similar to those used in the proof of Lemma 1, it is easy to see that both OBR and MBR belong to NP. We now prove the NP-hardness of MBR by means of a polynomial-time reduction from the acyclic subgraph problem. Then we further reduce MBR to our original realizability problem OBR. We recall that the acyclic subgraph problem is stated as follows:

Acyclic subgraph (in short, AS): given a directed graph $G = (V, A)$ and a positive integer h, does a set $A' \subseteq A$ exist such that $|A'| \geq h$ and the graph $G' = (V, A')$ is acyclic?

This problem has been proved to be NP-complete in [6].

Lemma 4. *One-sided bipartite realizability with mixed constraints is NP-hard.*

Proof. Given an instance of AS, we build an instance of MBR as follows. The bipartite graph $B = (V_0, V_1, E)$ is such that: (a) $V_1 = V$; (b) V_0 contains a distinct vertex x' for each vertex $x \in V$, i.e., $V_0 = \{x' \; s.t. \; x \in V\}$; (c) $\forall (x, y) \in A$ both (x', y) and (y', x) belong to E. Permutation π_0 is any ordering of the vertices in V_0. For each arc $(x, y) \in A$ we set up a constraint $(x', y; y', x)$: if $\pi_0(x') < \pi_0(y')$ we add $(x', y; y', x)$ to I, otherwise we add $(x', y; y', x)$ to C. $k = h$. Figure 1 provides an example of this reduction. A directed graph G is given in Figure 1(a) and the bipartite graph B obtained from it is shown in Figure 1(b), together with the whole set of constraints obtained choosing $\pi_0 = \langle d', a', c', b' \rangle$.

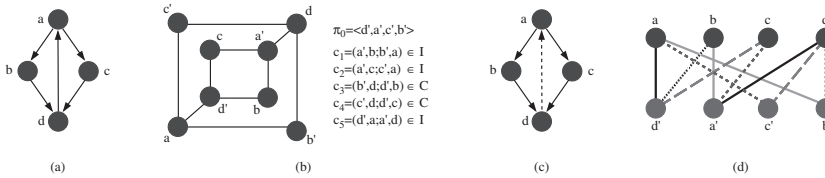


Fig. 1. From acyclic subgraph to one-sided realizability with mixed constraints.

It is worth observing that there is a one-to-one correspondence between constraints and arcs of the digraph and that the constraint $(x', y; y', x)$ associated to arc (x, y) belongs to C if and only if $\pi_0(x') > \pi_0(y')$. To prove the correspondence between solutions of AS and solutions of MBR we now exploit a basic idea: any permutation of the vertices of a digraph partitions its arcs into left-to-right and right-to-left arcs. We recall that a topological sort of an acyclic digraph yields no right-to-left arc. Based on this idea, Figure 2 points out the meaning of both crossing and non-crossing constraints: if arc (x', y') is left-to-right according to π_0, then we force x and y to have the same ordering as x' and y' using a crossing constraint (see Figure 2(a)); if arc (x', y') is right-to-left according to π_0, then we force x and y to have the opposite ordering of x' and y' using a non-crossing constraint (see Figure 2(b)).

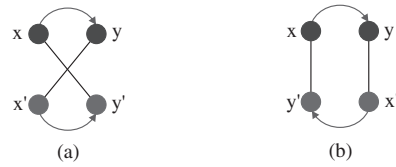


Fig. 2. Crossing and non-crossing constraints: (a) $(x', y; y', x) \in I$; (b) $(x', y; y', x) \in C$. In both cases arc (x, y) is forced to be a left-to-right arc.

Roughly speaking, our constraints force the presence of as many left-to-right arcs as possible. Proving that any permutation π_1 realizes k constraints from $I \cup C$ iff it yields an acyclic subgraph of G containing $h = k$ arcs is now easy. Let $c = (x', y; y', x)$ be a constraint from $I \cup C$. If $c \in I$ (i.e., $\pi_0(x') < \pi_0(y')$), then c is realized by π_1 if and only if $\pi_1(x) < \pi_1(y)$. Analogously, if $c \in C$ (i.e., $\pi_0(x') > \pi_0(y')$), then c is realized by π_1 if and only if $\pi_1(x) < \pi_1(y)$. In both cases the satisfaction of constraint c implies that arc (x, y) is a left-to-right arc according to π_1, and vice-versa. Therefore, any permutation π_1 satisfies a number of constraints equal to the number of left-to-right arcs that it induces.

In the example in Figure 1, choosing e.g. $\pi_1 = \langle a, b, c, d \rangle$ partitions the arcs of G as in Figure 1(c) and yields the bipartite straight-line drawing of B shown in Figure 1(d). In this drawing only constraint $(d', a; a', d) \in I$ is not satisfied; note this constraint corresponds to the unique dotted arc $(d, a) \in A \setminus A'$.

Theorem 2. *One-sided bipartite realizability is NP-hard.*

Proof. Let us consider an instance of MBR specified by a bipartite graph $B = (V_0, V_1, E)$, a set of crossing and non-crossing constraints $I \cup C$, a permutation π_0, and an integer k (w.l.o.g. in the following we assume that $\pi_0(a) < \pi_0(b)$ for any constraint $(a, x; b, y) \in I \cup C$). We build a corresponding instance of OBR specified by a bipartite graph $B' = (V_0', V_1', E')$, a set of non-crossing constraints C', a permutation π_0', and an integer k' as follows.

For each crossing constraint $(a, x; b, y) \in I$ we create a vertex named $\hat{ab}$, an edge $(\hat{ab}, y)$, and a non-crossing constraint $(\hat{ab}, y; a, x)$. We denote the sets of all the new vertices, the new edges, and the new constraints with $\hat{V_0}$, $\hat{E}$, and $\hat{C}$, respectively. We then set $V_0' = V_0 \cup \hat{V_0}$, $V_1' = V_1$, $E' = E \cup \hat{E}$, $C' = C \cup \hat{C}$, and $k' = k$. At last, we choose any permutation π_0' such that all vertices in $\hat{V_0}$ are to the left of vertices in V_0 (their relative positions are not important for our purposes), while the vertices in V_0 retain the same ordering as in π_0.

In the following we show that any permutation π_1 satisfies the same number of constraints in the instance of MBR and in the corresponding instance of OBR, proving a one-to-one correspondence between the solutions of the two problems. It is obvious that any permutation π_1 satisfies the same constraints from C in both instances. Moreover, there is a one-to-one correspondence between constraints in I and constraints in $\hat{C}$: let $(a, x; b, y) \in I$ and let $(\hat{ab}, y; a, x) \in \hat{C}$ be the corresponding non-crossing constraint. Recall that by construction we

have $\pi'_0(\hat{ab}) < \pi'_0(a) < \pi'_0(b)$. Therefore, the crossing constraint $(a, x; b, y)$ is realized by π_1 if and only if the non-crossing constraint $(\hat{ab}, y; a, x)$ is, proving that each satisfied constraint from I corresponds to a satisfied constraint from $\hat{C}$ and vice-versa.

Approximability and polynomiality. In order to prove that OBR is polynomial if $k = |C|$ and that a maximum set of realizable constraints can be approximated within a constant ratio we use an approximation preserving reduction [1] to the maximum acyclic subgraph problem. This problem is the optimization version of AS: given a digraph $H = (N, A, w)$ with positive arc weights w, it requires to find a maximum weight set of arcs $A' \subseteq A$ such that the subgraph $H' = (N, A')$ is acyclic.

Let $B = (V_0, V_1, E)$ and π_0 specify an instance of one-sided bipartite realizability. W.l.o.g. we assume that for each $(a, x; b, y) \in C$ it holds $\pi_0(a) < \pi_0(b)$; otherwise we change C by removing $(a, x; b, y)$ and adding the analogous constraint $(b, y; a, x)$. In the reduction we build a weighted *constraint digraph* $H = (N, A, w)$ as follows: $N = V_1$; $A = \{(x, y) : \exists\, a, b \in V_0 \; s.t. \; (a, x; b, y) \in C\}$; $\forall (x, y) \in A$, $w(x, y)$ is the number of constraints of C containing x and y in the second and fourth positions, respectively. In other words, the weight of arc (x, y) is the number of constraints that require $\pi_1(x) < \pi_1(y)$ in order to be realized.

An almost trivial observation is that H is acyclic iff all the constraints in C are realizable. This means that OBR can be solved in linear time if $k = |C|$, thanks to the fact that both the previous reduction and checking the acyclicity of the constraint digraph can be accomplished in linear time.

The following property also holds: each $A' \subseteq A$ such that the subgraph $H' = (N, A')$ is acyclic corresponds to a realizable set of constraints $C' \subseteq C$ with $|C'| \geq w(A')$. Indeed, if π_1 is a topological sort of H', then π_1 satisfies at least all the constraints corresponding by construction to the arcs in A', whose quantity is $w(A') = \sum_{(x,y)\in A'} w(x, y)$ (it could happen that π_1 satisfies more constraints if A' is not maximal). Finally, it is easy to see that the weight of a maximum acyclic subgraph of the constraint digraph equals the maximum number of realizable constraints, i.e., $w(A^*) = |C^*|$.

Let $r \geq 1$ be any approximation ratio for the maximum acyclic subgraph problem. From the previous considerations we have:

$$|C'| \geq w(A') \geq \tfrac{1}{r} \cdot w(A^*) = \tfrac{1}{r} \cdot |C^*|$$

proving that C' is an r-approximate solution for one-sided bipartite realizability, as well. Hence, a maximum set of realizable constraints can be approximated with ratio 2 [7].

4 Constrained Crossing Minimization

A widely used approach for creating layered drawings of directed graphs is presented in [14] and is known in literature as *hierarchical approach*. In order to produce readable drawings, different aesthetic criteria are taken into account by

this method; among them the number of edge crossings deserves special attention. In this section we show how to extend the hierarchical approach in order to support non-crossing constraints. Our extension hinges upon the approximation algorithm for one-sided bipartite realizability presented in Section 3. We first briefly recall how Sugiyama's algorithm works. Given a directed graph G, it performs four main steps. (1) *Cycle removal*: the direction of some arcs is temporarily reversed to make G acyclic. (2) *Layer assignment*: each vertex is assigned with a level so that all the arcs "flow" in the same direction. Dummy vertices are added for arcs spanning more than two levels. (3) *Crossing reduction*: vertices in each level are ordered so as to minimize the total number of edge crossings. (4) *Coordinates assignment*: vertical coordinates are proportional to layers, horizontal coordinates depend on the permutation of the vertices within each layer. At this point the direction of the reversed arcs is also restored.

In order to extend the hierarchical approach to support non-crossing constraints we need to focus on steps 2 and 3, because neither the removal of cycles nor the assignment of coordinates are affected by this kind of constraints. In the following we denote with G the directed graph to be drawn and with C the set of non-crossing constraints to be realized in the drawing.

Extending step 2. We recall that in this step the directed graph G is transformed into a proper layered graph. The set of constraints C must be therefore changed accordingly, due to the fact that dummy vertices could be added to split an edge involved in some constraint. We let the algorithm assign layers to vertices as usual, where $l(v)$ denotes the level of a vertex v after the layer assignment. Then we change C as follows.

Let (a, x) and (b, y) be any two arcs of G such that constraint $(a, x; b, y) \in C$; w.l.o.g. we can assume $l(a) < l(x)$ and $l(b) < l(y)$. If $l(x) = l(a) + 1$ and $l(y) = l(b)+1$ we do not touch C. Otherwise, at least a dummy vertex has been added to split (a, x) or (b, y) or both and constraint $(a, x; b, y)$ must be suitably replaced. Let $\langle u_k, u_{k+1}, \ldots, u_{k+s}\rangle$ be the path that replaced arc (a, x), with $u_k = a$, $u_{k+s} = x$, $k = l(a)$, and $k + s = l(x)$. Analogously, let $\langle v_h, v_{h+1}, ..., v_{h+t}\rangle$ be the path that replaced arc (b, y), with $v_h = b$, $v_{h+t} = y$, $h = l(b)$, and $h+t = l(y)$. Let also $[r, r+q] = [k, k+s] \cap [h, h+t]$. If $[r, r+q] \neq \emptyset$ we replace constraint $(a, x; b, y)$ with the constraints $(u_r, u_{r+1}; v_r, v_{r+1}) \ldots (u_{r+q-1}, u_{r+q}; v_{r+q-1}, v_{r+q})$. Otherwise we remove it, since it is going to be satisfied by any drawing obtained from layer assignment l. Figure 3 shows an example of the two cases. It is easy to get convinced that an original constraint will be realized in the final drawing if and only if all the new constraints are satisfied.

Thanks to the preprocessing of set C described above, we can now assume to deal only with constraints involving edges that join vertices in consecutive layers. More formally, we assume that G has k layers, named $L_0 \ldots L_k$, and that each $(a, x; b, y) \in C$ is such that $\exists i \in [0..k-1]$ such that $a, b \in L_i$ and $x, y \in L_{i+1}$. Based on this assumption, we can partition C into $k-1$ sets, named $C_{0,1} \ldots C_{k-1,k}$, where $C_{i,i+1}$ contains the constraints related to edges with endpoints in L_i and L_{i+1}.

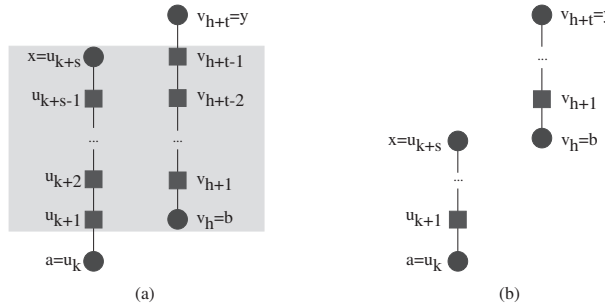


Fig. 3. Constraints replacement during the layer assignment step: dummy vertices are squared. (a) $(a, x; b, y)$ is replaced by constraints between edges in the grey band; (b) the grey band is empty: $(a, x; b, y)$ is removed from C.

Extending step 3. Before attempting at minimizing edge crossings, we introduce a *constraint realization* step, aimed at finding a large set of realizable constraints. Then we devise a crossing minimization strategy able to take into account and satisfy the set of realizable constraints previously identified.

We recall that the most used strategy for crossing minimization in hierarchical drawings is based on a layer-by-layer sweep heuristic [3], that requires to repeatedly solve a crossing minimization problem on a bipartite graph with one side fixed. We therefore adapt the layer-by-layer sweep method to support non-crossing constraints, by repeatedly solving a one-sided bipartite realizability problem as follows.

A vertex ordering π_0 for L_0 is first randomly chosen. Then, for $i = 0$ to $k-1$, the bipartite subgraph B_i of G induced by the vertices in $L_i \cup L_{i+1}$ and the set of constraints $C_{i,i+1}$ are considered. An approximated solution to OBR on this instance is found, keeping layer L_i fixed. This leads to identify a set $C'_{i,i+1} \subseteq C_{i,i+1}$ of realizable constraints. We have then to find a vertex ordering for L_{i+1} which minimizes edge crossings while realizing the constraints of $C'_{i,i+1}$. Not all crossing minimization algorithms can be easily adapted to accomplish this task (think, e.g., of the well-known median/barycenter algorithms [4,14]); here we suggest a modification of the penalty minimization technique discussed in [2].

With this method, a penalty digraph P (with vertex set L_{i+1}) is built from B_i and π_i, a feedback arc set F is found on P, and a topological sort of the vertices in the subgraph of P obtained by deleting arcs in F is returned as ordering for L_{i+1}. We recall that a feedback arc set of a directed graph is a subset of arcs whose removal makes the digraph acyclic; the feedback arc set problem requires to find a minimum weight feedback arc set.

Crossing constraints can be incorporated in the penalty based approach by suitably assigning weights to some arcs of P and by applying the previous algorithm as is. Namely, if $(a, x; b, y)$ is a constraint in $C'_{i,i+1}$ with $a, b \in L_i$ and $x, y \in L_{i+1}$, we add arc (x, y) to P if it does not already exist, and we set

$w(x, y) = \infty$. Since $C'_{i,i+1}$ is realizable, the subgraph of P induced by the infinite weight arcs is acyclic. Any approximation algorithm for the feedback arc set problem applied on P will therefore choose none of these arcs, thus minimizing crossings while satisfying all the constraints in $C'_{i,i+1}$.

As far as the algorithm above is concerned, it may happen that a pair of edges decomposed by means of dummy vertices may cross several times. This can be avoided by assigning ∞ weight to suitably chosen arcs in the constraint digraph. We also remark that our algorithm can be extended to deal with constraints concerning the relative positions of vertices within each layer. Due to the lack of space, we defer the details to the full paper.

References

1. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Approximate Solution of NP-Hard Optimization Problems.* Springer Verlag, 1999.
2. Demetrescu, C. and Finocchi, I. Break the "right" cycles and get the "best" drawing. In B.E. Moret and A.V. Goldberg, editors, *Proc. 2nd Int. Workshop on Algorithm Engineering and Experiments (ALENEX'00)*, 171–182, 2000.
3. G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, Upper Saddle River, NJ, 1999.
4. P. Eades and N.C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.
5. E.R. Gansner, E. Koutsofios, S.C. North, and K.P. Vo. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 19:214–230, 1993.
6. M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to Theory of NP-completeness.* W.H.Freeman, 1979.
7. R. Hassin and S. Rubinstein. Approximations for the maximum acyclic subgraph problem. *Information Processing Letters*, 51:133–140, 1994.
8. S. Leipert. *Level Planarity Testing and Embedding in Linear Time.* PhD-Thesis, Universität zu Köln, 1998.
9. J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.
10. J. Kratochvíl. String graphs II: Recognizing string graphs is NP-hard. *Journal Combin. Theory Ser. B*, 52:67–78, 1991.
11. J. Kratochvíl. Crossing number of abstract topological graphs. In *Proc. 6th Int. Symp. on Graph Drawing (GD'98)*, LNCS 1547, 238–245, 1998.
12. J. Kratochvíl, A. Lubiw, and J. Nesetril. Noncrossing subgraphs of topological layouts. *SIAM Journal on Discrete Mathematics*, 4:223–244, 1991.
13. J. Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979.
14. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125, 1981.

On the Validity of Hierarchical Decompositions*

Irene Finocchi and Rossella Petreschi

Department of Computer Science
University of Rome "La Sapienza"
{finocchi, petreschi}@dsi.uniroma1.it

Abstract. Hierarchical decompositions are a useful tool for drawing large graphs. Such decompositions can be represented by means of a data structure called hierarchy tree. In this paper we introduce the notion of $\mathcal{P}$-*validity* of hierarchy trees with respect to a given property $\mathcal{P}$: this notion reflects the similarity between the topological structure of the original graph and of any high-level representation of it obtained from the hierarchy tree. We study the $\mathcal{P}$-*validity* when the clustered graph is a tree and property $\mathcal{P}$ is the acyclicity, presenting a structure theorem for the $\mathcal{P}$-*validity* of hierarchy trees under these hypotheses.

1 Introduction and Preliminaries

Many graph drawing algorithms have been designed during the last years [2] and most of them move from the following assumption: in order to find a pleasant layout of a graph, we first of all should be able to recognize its graph-theoretical properties, e.g., acyclicity, planarity, bipartiteness, and so on. Actually, exploiting the relevant features of a graph helps produce better visualizations for it. It is also well accepted that only few graph drawing algorithms scale up well and are able to visualize the more and more large graphs that arise in practical applications. Clustering techniques are a useful tool to overcome this drawback and make it possible dealing with graphs not fitting in the screen [3,4,5,6]. Recursively clustering the vertices of a graph leads to a hierarchical decomposition of the graph itself: each cluster in the decomposition is considered as a single node which is visualized instead of a set of vertices of the original graph, considerably reducing the dimension of the drawing. Thanks to the parent relationships between clusters in the hierarchical decomposition, the viewer can move from a high-level representation of the graph to another one by detailing or shrinking clusters. The considerations above immediately lead to the following problem: *Given a hierarchical decomposition of a graph G and a graph-theoretical property $\mathcal{P}$ satisfied by G, does any high-level representation of G obtained from the hierarchical decomposition satisfy property $\mathcal{P}$?* Though this question naturally arises when using hierarchical decompositions for visualizing large graphs, as far as we

* Work partially supported by the project "Algorithms for Large Data Sets: Science and Engineering" of the Italian Ministry of University and of Scientific and Technological Research (MURST 40%).

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 368–374, 2001.
© Springer-Verlag Berlin Heidelberg 2001

know it has not been previously studied in literature. Some related work is due to Feng *et al.*, who focus on clustered planar graphs and investigate the concept of compound planarity [5].

In this paper we introduce the general notion of $\mathcal{P}$-*validity* of hierarchical decompositions of graphs with respect to a given property $\mathcal{P}$. This notion reflects the similarity between the topological structure of the original graph and of any high-level representation of it and allows us to compare different hierarchical decompositions associated to the same graph. We then focus on the case where the clustered graph is a tree and property $\mathcal{P}$ is the acyclicity and we present a structural characterization of the $\mathcal{P}$-*validity* under these hypotheses, i.e., conditions on the structure of the clusters necessary and sufficient to guarantee the $\mathcal{P}$-*validity* of the decomposition.

We start with preliminary definitions and notation. A hierarchical decomposition of a graph can be represented by means of a data structure called *hierarchy tree* (or sometimes *cluster* or *inclusion* tree) [1,5]. A hierarchy tree $HT = (N, A)$ associated to a graph $G = (V, E)$ is a rooted tree whose leaves are the vertices of G. Nodes of a hierarchy tree are called *clusters*: a cluster c represents a set V_c of vertices of G, namely, the vertices that are the leaves of the subtree rooted at c. We say that such vertices are *covered* by c and we refer to their number as *cardinality* of c. For brevity, we write $u \prec c$ to indicate that a vertex $u \in V$ is covered by a cluster $c \in N$. We call *singleton cluster* a cluster with cardinality equal to 1. W.l.o.g. we assume that the vertices covered by a cluster c are a proper subset of the vertices covered by the parent of c in HT, i.e., in the hierarchy tree there is no node with a unique child. Under this hypothesis, the number of nodes of a hierarchy associated to a n-vertex graph is at most $2n - 1$. This implies that adding a hierarchy tree on the top of a graph requires only space linear in the number of vertices of the graph itself.

It is possible to visit a hierarchy tree in a top-down fashion by performing expand and contract operations on the clusters visualized at any instant of time: these operations define a *navigation* of HT. The previous ideas can be formalized by defining the concepts of *covering* and of *view* [1,6]:

Definition 1. *A covering C of a graph $G = (V, E)$ on a hierarchy tree $HT = (N, A)$ is a subset of the nodes of HT such that each vertex of G is covered by exactly a node in C.*

A view of graph G on HT is a graph $W = (C, L)$ such that C is a covering of G on HT and $L = \{(c, c') \mid c, c' \in C,\ c \neq c',\ \exists (u, v) \in E\ :\ u \prec c \text{ and } v \prec c'\}$.

We refer to the edges of W as *links*. It is worth observing that any view is a simple graph: if there exist two edges $(u, v) \in E$ and $(u', v') \in E$ which lead to a connection between two clusters c and c', only a single link (c, c') is considered in L. We also call W_r and $W_l = G$ the views generated by the root of the hierarchy tree and by all its leaves, respectively: in other words, W_r and W_l are the least and the most detailed representations of the graph, respectively.

2 $\mathcal{P}$-Validity of Views and Hierarchy Trees

A navigation of a hierarchy tree should help the viewer to focus on particular interesting regions of the graph, according to his/her necessities; hence, if the clusters are generated not taking into account the topological structure of the graph, no benefit may derive for the viewer from the clustering structure. Informally speaking, only vertices in the same "locality" of the graph should be grouped together to form a cluster.

A motivating example of the concept of $\mathcal{P}$-*validity* is illustrated in Figure 1, where two different hierarchical decompositions associated to a chain with 8 vertices are considered (Figure 1(a) and Figure 1(d)). The hierarchy trees HT_1 (Figure 1(b)) and HT_2 (Figure 1(e)) corresponding to these decompositions are both complete binary trees of height 3 and differ only in the permutation of their leaves. Figure 1(c) and Figure 1(f) report three views related to coverings $\{\mathtt{E},\mathtt{F}\}$, $\{\mathtt{E},\mathtt{C},\mathtt{D}\}$, and $\{\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{D}\}$ in the two decompositions, respectively. All the views built from HT_1 maintain the structural property of the original graph to be a chain, while the views from HT_2 loose this property introducing cycles, up to become even a clique.

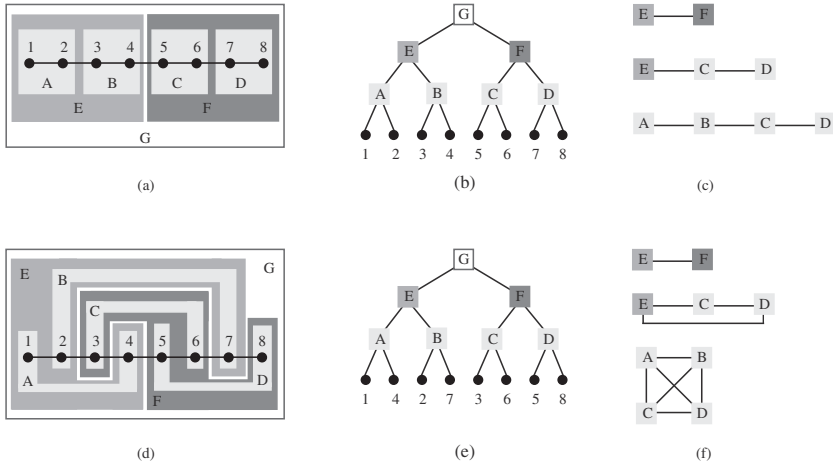


Fig. 1. Hierarchical decompositions, hierarchy trees, and views of a 8-vertex chain

In order to characterize the "semantic" differences of hierarchy trees with "syntactically" similar or even identical structure it is then natural to introduce a notion of validity of a view and of a hierarchy tree with respect to a certain property $\mathcal{P}$: from now on we refer to this concept as $\mathcal{P}$-*validity*.

Definition 2. *Let HT be a hierarchy tree associated to a graph G and let $\mathcal{P}$ be a property satisfied by G. A view W of G obtained from HT is $\mathcal{P}$-valid iff W*

satisfies property $\mathcal{P}$. HT is $\mathcal{P}$-valid iff all the views of G obtained from it are $\mathcal{P}$-valid.

In view of the fact that Definition 2 is parametric in the property $\mathcal{P}$ to be considered, different notions of validity may be thought for different classes of graphs. For example, we may require a view of a bipartite graph to be bipartite or a view of a planar graph to be planar. But even more sophisticated definitions for $\mathcal{P}$ can be considered: as an example, we refer to the *c-planarity* property introduced by Feng *et al.* [5]. The interest in characterizing $\mathcal{P}$-valid hierarchy trees naturally follows from the considerations in Section 1. In this paper we focus on the case where the clustered graph is a tree and property $\mathcal{P}$ is the acyclicity: for brevity, we will speak of valid views/hierarchy trees under these hypotheses. In Section 3 we prove necessary and sufficient conditions for a hierarchy tree to be valid.

3 A Structural Characterization of Valid Hierarchy Trees

Before presenting the structure theorem for the validity of a hierarchy tree, we give preliminary definitions and lemmas useful for proving it. First of all we study the connectivity of views obtained from hierarchy trees of connected graphs.

Lemma 1. *Each view obtained from a hierarchy tree associated to a connected graph is connected.*

In the following we denote with $S(c)$ the subgraph of $T = (V, E)$ induced by the vertices covered by a node c of HT. For each $u, v \in T$, let $path_T(u, v)$ be the path joining u and v in T and let $dist_T(u, v)$ be the length of this path.

Definition 3. *Let c be a node of a hierarchy tree HT associated to a free tree T. Let u and v be two vertices of T covered by c. u, v are a broken pair of cluster c iff they are neither coincident nor connected in $S(c)$. A broken pair u, v is a minimum-distance broken pair of c iff $w \not\prec c$, $\forall w \in path_T(u, v)$ such that $w \neq u, v$.*

Lemma 2. *Let W be a view on a hierarchy tree HT associated to a free tree T. If W is not acyclic, then in each cycle $\mathcal{C}$ there is at least a cluster containing a broken pair.*

Proof. Let $\mathcal{C} = (c_1, \ldots, c_h)$ be a cycle in W. Each cluster $c_i \in \mathcal{C}$ is endpoint of two links (c_{i-1}, c_i) and (c_i, c_{i+1}), respectively (to simplify the notation we assume that $h + 1 = 1$). Let us call (r_{i-1}, l_i) and (r_i, l_{i+1}) two tree edges which derive links (c_{i-1}, c_i) and (c_i, c_{i+1}), respectively. For each cluster c_i we therefore identify two vertices covered by it, named l_i and r_i (see Figure 2). If no broken pair exists in any cluster of $\mathcal{C}$, for each $i \in [1, h]$ vertices l_i and r_i are either coincident or connected in $S(c_i)$. This implies the existence of a cycle in tree T, that is a contradiction.

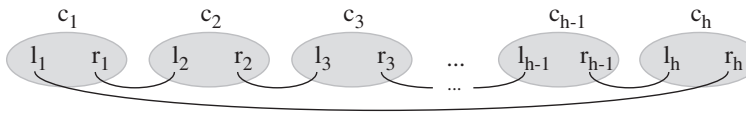


Fig. 2. Vertices, edges, clusters, and links involved in cycle $\mathcal{C}$ in the proof of Lemma 2

Theorem 1. *Let $T = (V, E)$ be a free tree and let $HT = (N, A)$ be a hierarchy tree associated to T. HT is valid iff for each minimum-distance broken pair u, v of HT $dist_T(u, v) = 2$.*

Proof. We first prove the necessary condition, showing that the existence in HT of a minimum-distance broken pair u, v such that $dist_T(u, v) > 2$ implies that HT is not valid. Let c be a node of HT such that u, v are a minimum-distance broken pair in c. Let us consider the view $W = (C, L)$ where covering C consists only of singleton clusters, except for cluster c: $C = \{c\} \cup \{\{x\},\ x \in V \text{ and } x \not\prec c\}$. Let z and w be the vertices in $path_T(u, v)$ such that $(u, z) \in E$ and $(w, v) \in E$. Since u, v is a minimum-distance broken pair of c, $z \not\prec c$, $w \not\prec c$, and any other vertex in $path_T(u, v)$ is not covered by c. Moreover, since $dist_T(u, v) > 2$, $dist_T(z, w) > 0$, i.e., z and w are distinct vertices. Hence $\langle c, \{z\}, \ldots, \{w\}, c\rangle$ is a cycle in W, proving that HT is not valid.

We now focus on the sufficient condition, proving that a contradiction can be derived if we suppose that HT is not valid while satisfying the property on the minimum-distance broken pairs formulated in the statement of the theorem. Namely, we assume that in HT there exists a non-valid view $W = (C, L)$, i.e., a view W that is not a tree. Since W must be connected due to Lemma 1, it must contain a simple cycle $\mathcal{C}$. We prove that this assumption leads to a contradiction.

The general idea of the proof is to convert W into another view W', still existing on the hierarchy tree, such that: (a) the number of singletons of W' is strictly greater than the number of singletons of W; (b) the cycle $\mathcal{C}$ of W is also changed into a simple cycle $\mathcal{C}'$ in W'. The sequence of manipulations that we perform has finite length, as the number of singletons is clearly upper bounded by $n = |V|$. We can therefore prove that at some step during this process we find a contradiction due to Lemma 2, since we obtain a cycle with no broken pairs.

Let $\mathcal{C} = (c_1, \ldots, c_h)$ be a simple cycle of length h in W and let c_i, l_i, and r_i be defined as in the proof of Lemma 2 for $1 \leq i \leq h$ (see also Figure 2). Due to Lemma 2 a broken pair must exist in $\mathcal{C}$. Let l_i and r_i be the vertices in such a broken pair of $\mathcal{C}$. We remark that l_i and r_i are not necessarily a minimum-distance broken pair.

Let us consider the path from l_i to r_i in T, which is unique and not completely contained in $S(c_i)$. On this path we can univocally identify a set of k vertices u_j, for $1 \leq j \leq k$, such that $u_j \prec c_i$ but its successor on $path_T(l_i, r_i)$ is not covered by c_i. Analogously, we can univocally identify a set of k vertices v_j, for $1 \leq j \leq k$, such that $v_j \prec c_i$ but its predecessor on $path_T(l_i, r_i)$ is not covered

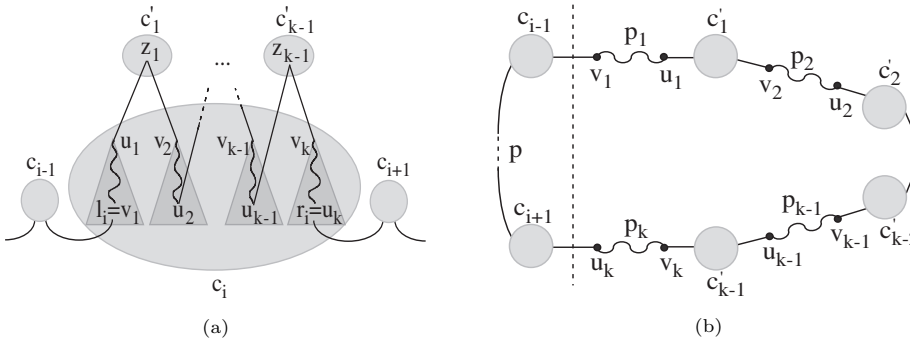


Fig. 3. (a) Broken pair l_i, r_i in cycle $\mathcal{C}$; (b) cycle $\hat{\mathcal{C}}$ in the proof of Theorem 1

by c_i. Observe that it could be $u_j = v_j$ for some j, but vertices with different indexes always belong to different connected components of $S(c_i)$.

It is clear that $v_1 = l_i$ and $u_k = r_i$. Besides, it is easy to see that the pairs u_j, v_{j+1}, for $1 \leq j \leq k-1$, are minimum distance broken pairs of cluster c_i and, by hypothesis, we know that $dist_T(u_j, v_{j+1}) = 2$. Let us call z_j the unique vertex of T in the path between u_j and v_{j+1} and let c'_j be the cluster of W that covers z_j. The configuration is illustrated in Figure 3(a).

We now change view W into W' by expanding cluster c_i at the singleton level, i.e., by substituting c_i with the set of singleton clusters corresponding to the vertices in $S(c_i)$. In the following we prove that we are able to exhibit in W' a simple cycle $\mathcal{C}'$. Let us first consider the cycle $\hat{\mathcal{C}}$ shown in Figure 3(b). $\hat{\mathcal{C}}$ clearly exists in W', being obtained from $\mathcal{C}$ by unrolling $path_T(l_i, r_i)$ (compare vertices and clusters involved in Figure 3(a) and in Figure 3(b), respectively). However, $\hat{\mathcal{C}}$ is not necessarily simple. W.l.o.g. we can assume that $c'_j \neq c'_s \ \forall j, s \in [1, k-1]$, $j \neq s$. We can always reduce to this situation as follows: while $\hat{\mathcal{C}}$ contains a pair j, s such that $1 \leq j < s \leq k-1$ and $c'_j = c'_s$, substitute the path $\langle c'_j, \ldots, c'_s \rangle$ with the path $\langle c'_j \rangle$. After this operation all the c'_j are distinct. If $\forall j \in [1, k-1]$ $c'_j \notin \mathcal{C}$, $\mathcal{C}' = \hat{\mathcal{C}}$ is a simple cycle. Otherwise $\exists j \in [1, k-1]$ such that $c'_j \in \mathcal{C}$ and two cases may happen:

- $c'_1 = c_{i-1}$: change $\hat{\mathcal{C}}$ by replacing the non-simple path $\langle c_{i-1}, p_1, c'_1, p_2 \rangle$ with the simple path $\langle c_{i-1}, p_2 \rangle$. If $k = 1$ or $\forall j \in [2, k-1]$ $c'_j \notin \mathcal{C}$, then the modified $\hat{\mathcal{C}}$ is a simple cycle and we can choose $\mathcal{C}' = \hat{\mathcal{C}}$. Otherwise, let s be the smallest value in $[2, k-1]$ such that $c'_s \in \mathcal{C}$, i.e., $c'_s = c_t$ for some $t \in [1, h]$, $t \neq i$. Moving clockwise on $\hat{\mathcal{C}}$, we find the simple cycle $\mathcal{C}' = \langle c_{i-1}, p_2, c'_2, \ldots, p_s, c'_s = c_t \sim c_{i-1} \rangle$, where $\sim$ indicates a subpath of p.

- $c'_1 \neq c_{i-1}$: let s be the smallest value in $[1, k-1]$ such that $c'_s \in \mathcal{C}$, i.e., $c'_s = c_t$ for some $t \in [1, h]$, $t \neq i$. If $s = 1$ then $\mathcal{C}' = \langle c_{i-1}, p_1, c'_1 = c_t \sim c_{i-1} \rangle$ is a simple cycle in W'. Note that the length of the path $\langle c_t \sim c_{i-1} \rangle$ is ≥ 1, because $c_t = c'_1 \neq c_{i-1}$. If $s > 1$ then let $\mathcal{C}' = \langle c_{i-1}, p_1, c'_1, p_2, c'_2, \ldots, p_s, c'_s = c_t \sim c_{i-1} \rangle$.

In this case it could be $c_t = c_{i-1}$; however, the fact that the path from c_{i-1} to c'_s has length ≥ 3 guarantees $\mathcal{C}'$ to be a cycle of W'. By construction we also know that $\mathcal{C}'$ is simple.

In any case we are therefore able to find a view W' containing more singletons than W and to exhibit a simple cycle $\mathcal{C}'$ in W'. Iterating this reasoning, we obtain either a cycle with no broken pairs, which is absurd due to Lemma 2, or a cyclic view containing only singleton clusters, i.e., W_l. This is also a contradiction because W_l is acyclic being equal to T.

Theorem 1 implies that the validity of a hierarchy tree can be checked in polynomial time. We remark that this does not immediately follow from Definition 2, since the number of views in a hierarchy tree may be exponential.

References

1. Buchsbaum, A.L., and Westbrook, J.R.: *Maintaining hierarchical graph views*, Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA'00), 566-575, 2000.
2. Di Battista, G., Eades, P., Tamassia, R., and Tollis, I.: *Graph Drawing: Algorithms for the visualization of graphs*, Prentice Hall, Upper Saddle River, NJ, 1999.
3. Duncan, C.A., Goodrich, M.T., and Kobourov, S.G.: *Balanced aspect ratio trees and their use for drawing large graphs*, Journal of Graph Algorithms and Applications, 4(3), 19-46, 2000.
4. Eades, P., and Feng, Q.: *Multilevel visualization of clustered graphs*, Proc. 4th Symposium on Graph Drawing (GD'96), LNCS 1190, 101-112, 1997.
5. Feng, Q.W., Cohen, R.F., and Eades, P.: *Planarity for clustered graphs*, Proc. European Symposium on Algorithms (ESA'95), LNCS 979, 213-226, 1995.
6. Finocchi, I., and Petreschi, R.: *Hierarchical clustering of trees: algorithms and experiments*, Proc. 3rd Workshop on Algorithm Engineering and Experiments (ALENEX'01), January 2001.

Lower Bounds on the Minus Domination and k-Subdomination Numbers

Liying Kang[1], Hong Qiao[2], Erfang Shan[3], and Ding-Zhu Du[4,5]

[1] Dept. of Mathematics, Shanghai University, Shanghai 200436, China
[2] Dept. of Manufacturing Engineering and Engineering Management, City University of Hong Kong, Hong Kong, China
[3] Dept. of Mathematics, Shijiazhuang Normal College, Shijiazhuang 050041, China
[4] Dept. of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA
[5] Acamemy of Mathematics and System Sciences, Chinese Academy of Sciences, Beijing 100080, China.

Abstract. A three-valued function f defined on the vertex set of a graph $G = (V, E)$, $f : V \to \{-1, 0, 1\}$ is a *minus dominating function* if the sum of its function values over any closed neighborhood is at least one. That is, for every $v \in V$, $f(N[v]) \geq 1$, where $N[v]$ consists of v and all vertices adjacent to v. The weight of a minus function is $f(V) = \sum_{v \in V} f(v)$. The minus domination number of a graph G, denoted by $\gamma^-(G)$, equals the minimum weight of a minus dominating function of G. In this paper, sharp lower bounds on minus domination of a bipartite graph are given. Thus, we prove a conjecture proposed by J. Dunbar etc.(Discrete Math. 199(1999) 35-47), and we give a lower bound on $\gamma_{ks}(G)$ of a graph G.

1 Introduction

For a graph $G = (V, E)$ with vertex set V and edge set E, the *open neighborhood* of $v \in V$ is $N(v) = \{u \in V | uv \in E\}$ and the *closed neighborhood* of v is $N[v] = \{v\} \cup N(v)$. For a set S of vertices, we define the open neighborhood $N(S) = \cup_{v \in S} N(v)$, and the closed neighborhood $N[S] = N(S) \cup S$. A *dominating set* S for a graph $G = (V, E)$ is a subset of the vertex set V such that every vertex $v \in V$ is either in S or adjacent to a vertex in S. The *domination number* of G, $\gamma(G)$, equals the minimum cardinality of a dominating set.

For a real function f defined on vertices of a graph G and $S \subseteq V$, write $f(S) = \sum_{v \in S} f(v)$ and $f[v] = f(N[v])$. A *minus dominating function* of G is defined in [3] as a function $f : V \to \{-1, 0, 1\}$ such that $f[v] \geq 1$ for each $v \in V$. A *signed dominating function* of G is defined in [5] as $f : V \to \{-1, 1\}$ satisfying $f[v] \geq 1$ for all $v \in V$. A minus(signed) dominating function f is minimal if every minus(signed) dominating function g satisfying $g(v) \leq f(v)$ for every $v \in V$, is equal to f. It is easy to see that a minus dominating function is minimal if and only if for every vertex $v \in V$ with $f(v) \geq 0$, there exists a vertex $u \in N[v]$ with $f[u] = 1$ and a signed function is minimal if and only if every vertex v of weight 1, there exists some $u \in N[v]$

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 375–383, 2001.
© Springer-Verlag Berlin Heidelberg 2001

such that $f[u] = 1$ or 2. The *minus domination number* for a graph G is $\gamma^-(G) = \min\{f(V)|\ f$ is a minimal minus dominating function$\}$. Likewise the *signed domination number* for a graph G is $\gamma_s(G) = \min\{f(V)|\ f$ is a minimal signed dominating function$\}$

A *majority dominating function* of G is define in [2] as $f : V \to \{-1, 1\}$ such that $f[v] \geq 1$ for at least half the vertices of G, and the minimum weight of such a function is the *majority domination number*.

For a positive integer k, a *k-subdominating function*(kSF) of G is a function $f : V \to \{-1, 1\}$ such that $f[v] = \sum_{u \in N(v)} f(u) \geq 1$ for at least k vertices of G. The *aggregate ag(f)* of such a function is defined by $ag(f) = \sum_{v \in V} f(v)$ and the *k-subdomination number* $\gamma_{ks}(G)$ by $\gamma_{ks} = min\{ag(f) : f$ is a kSF of $G\}$. In the special cases $k = |V|$ and $k = \lceil \frac{|V|}{2} \rceil$, γ_{ks} is respectively the signed domination number $\gamma_s(G)$ and the majority domination number $\gamma_{maj}(G)$.

Since the problems of determining the signed domination number and minus domination number are NP-complete, many works on bounds for $\gamma^-(G)$ and $\gamma_s(G)$ were studied in [4, 7, 8, 9, 10, 11, 12]. In [3], the following conjecture was given.

Conjecture 1 [3]. If G is a bipartite graph of order n, then $\gamma^-(G) \geq 4(\sqrt{n+1} - 1) - n$.

2 Lower Bounds on Minus Domination of a Bipartite Graph

Theorem 1. If $G = (X, Y)$ is a bipartite graph, then $\gamma^-(G) \geq 4(\sqrt{n+1} - 1) - n$

Proof. Let f be a minus dominating function of G satisfying $f(V) = \gamma^-(G)$ and

$$M = \{v \in V|\ f(v) = -1\}$$
$$P = \{v \in V|\ f(v) = 1\}$$
$$Q = \{v \in V|\ f(v) = 0\}$$

$M_X = M \cap X,\ M_Y = M \cap Y,\ P_X = P \cap X,\ P_Y = P \cap Y,\ Q_X = Q \cap X,\ Q_Y = Q \cap Y,\ m_x = |M_X|,\ m_y = |M_Y|,\ p_x = |P_X|,\ p_y = |P_Y|,\ q_x = |Q_X|,\ q_y = |Q_Y|$.

Since $f[v] \geq 1$ for every $v \in V$, we have $|N(v) \cap P_X| \geq 2$ for every $v \in M_Y$. So

$$e(P_X, M_Y) \geq 2m_y. \tag{1}$$

For every $v \in P_X$, $|N(v) \cap M_Y| \leq |N(v) \cap P_Y|$. Then

$$\begin{aligned} e(P_X, M_Y) &= \sum_{v \in P_X} |N(v) \cap M_Y| \\ &\leq \sum_{v \in P_X} |N(v) \cap P_Y| \\ &\leq p_x p_y. \end{aligned} \tag{2}$$

By (1) and (2) we have

$$2m_y \leq p_x p_y.$$

Similarly

$$2m_x \leq p_x p_y,$$

then

$$m_x + m_y \leq p_x p_y. \tag{3}$$

Since

$$n = q_x + q_y + m_x + m_y + p_x + p_y,$$

and

$$2\sqrt{p_x p_y} \leq p_x + p_y,$$

we have

$$2\sqrt{p_x p_y} + m_x + m_y + q_x + q_y \leq n. \tag{4}$$

Using (3) and (4) we have

$$2\sqrt{m_x + m_y} + m_x + m_y + q_x + q_y \leq n, \tag{5}$$

and

$$2\sqrt{m_x + m_y} + m_x + m_y \leq n. \tag{6}$$

Using (5) we obtain

$$\begin{aligned}\gamma^-(G) &= f(V(G)) \\ &= n - (q_x + q_y) - 2(m_x + m_y) \\ &\geq m_x + m_y + 2\sqrt{m_x + m_y} - 2(m_x + m_y) \\ &= 2\sqrt{m_x + m_y} - (m_x + m_y). \end{aligned} \tag{7}$$

For notation convenience, we define the following

$$\begin{aligned} a &= \sqrt{m_x + m_y}, \\ h(y) &= y^2 + 2y \quad (y \geq 1), \\ g(y) &= 2y - y^2 \quad (y \geq 1). \end{aligned}$$

Since $\frac{dh}{dy} = 2y + 2 \geq 2$, $\frac{dg}{dy} = 2 - 2y \leq 0$, so $h(y)$ is a monotonous increasing function and $g(y)$ is a monotonous decreasing function. By (6) we have $h(a) = a^2 + 2a \leq n$. And when $y = -1 + \sqrt{1+n}$,

$$\begin{aligned} h(y) &= (-1 + \sqrt{1+n})^2 + 2(-1 + \sqrt{1+n}) \\ &= 1 - 2\sqrt{1+n} + 1 + n - 2 + 2\sqrt{1+n} \\ &= n. \end{aligned}$$

So $a \leq -1 + \sqrt{1+n}$.

By (7) we obtain

$$\begin{aligned}\gamma^-(G) &\geq g(a)\\ &\geq g(-1+\sqrt{1+n})\\ &= 2(-1+\sqrt{1+n}) - (-1+\sqrt{1+n})^2\\ &= 2(-1+\sqrt{1+n}) - (1-2\sqrt{1+n}+1+n)\\ &= 4(\sqrt{n+1}-1) - n.\end{aligned}$$

We now show that this bound is best possible by the following graphs G construct by J. Dunbar ect.[3]. Let $s \geq 4$ be an even integer, and let H be isomorphic to $\frac{s}{2}$ disjoint copies of $K_{2,s}$. Let H_1 and H_2 be two disjoint copies of H. Further, let X_i and Y_i be the sets of vertices of H_i of degree 2 and s, respectively, for $i = 1, 2$. Now let G be the graph obtained from $H_1 \cup H_2$ by joining every vertex of Y_1 to every vertex of Y_2. Then G is a bipartite graph of order $n = s(s+2)$ with partite sets $X_1 \cup Y_2$ and $X_2 \cup Y_1$. Let f be the function on G defined as follows: let $f(v) = -1$ if $v \in X_1 \cup X_2$, and let $f(v) = 1$ if $v \in Y_1 \cup Y_2$. Then it is easy to verify that f is a minus dominating function on G with $\gamma^-(G) = f(V(G)) = 2s - s^2 = 4(\sqrt{n+1}-1) - n$. □

Theorem 2. If $G = (X, Y)$ is a bipartite graph of order n, then

$$\gamma^-(G) \geq \lceil n - (\frac{\epsilon}{\delta} + \frac{\epsilon}{1 + max(\delta_X, \delta_Y)}) \rceil$$

where $\delta_X = min\{d(v)|v \in X\}, \delta_Y = min\{d(v)|v \in Y\}$, and the bound is sharp.

Proof. Let $f, M_X, M_Y, P_X, P_Y, Q_X, Q_Y, m_x, m_y, q_x, q_y, p_x$ and p_y be defined as in the proof of Theorem 1. For any $x \in V$, let t_x denotes the number of vertices of weight 0 in $N(x)$. Then we have

$$|N(x) \cap M| \leq \begin{cases} \frac{d(x)-t_x}{2}, & if\ x \in P \\ \frac{d(x)-1-t_x}{2}, & if\ x \in Q, \\ \frac{d(x)-t_x}{2} - 1 & if\ x \in M \end{cases}$$

So

$$\begin{aligned}\sum_{y \in M_Y} d(y) &= \sum_{x \in P_X} |N(x) \cap M_Y| + \sum_{x \in Q_X} |N(x) \cap M_Y| + \sum_{x \in M_X} |N(x) \cap M_Y|\\ &\leq \sum_{x \in P_X} \frac{d(x) - t_x}{2} + \sum_{x \in Q_X} \frac{d(x) - 1 - t_x}{2} + \sum_{x \in M_X} (\frac{d(x) - t_x}{2} - 1)\\ &= \sum_{x \in X} (\frac{d(x)}{2} - \frac{t_x}{2}) - \frac{1}{2} q_x - m_x \qquad (8)\end{aligned}$$

Obviously

$$m_y \delta_Y \leq \sum_{y \in M_Y} d(y) \qquad (9)$$

$$\sum_{x\in X}\frac{t_x}{2}=\frac{1}{2}\sum_{y\in Q_Y}d(y)\geq\frac{1}{2}\delta_Y q_y. \tag{10}$$

Combining (8), (9) and (10) we obtain

$$(q_y+2m_y)\delta_Y+(q_x+2m_x)\leq\epsilon. \tag{11}$$

Similarly we have

$$(q_x+2m_x)\delta_X+(q_y+2m_y)\leq\epsilon. \tag{12}$$

If $q_x+2m_x\leq q_y+2m_y$, by (11) and (12) we have

$$\begin{aligned}q_x+2m_x&\leq\frac{\epsilon}{1+max(\delta_X,\delta_Y)},\\ q_y+2m_y&\leq\frac{\epsilon}{\delta_Y}.\end{aligned}$$

So

$$\begin{aligned}\gamma^-(G)&=n-(q_x+2m_x+q_y+2m_y)\\ &\geq\lceil n-(\frac{\epsilon}{\delta_Y}+\frac{\epsilon}{1+max(\delta_X,\delta_Y)})\rceil\\ &\geq\lceil n-(\frac{\epsilon}{\delta}+\frac{\epsilon}{1+max(\delta_X,\delta_Y)})\rceil.\end{aligned}$$

If $q_y+2m_y<q_x+2m_x$, by (11) and (12) we have

$$\begin{aligned}q_y+2m_y&<\frac{\epsilon}{1+max(\delta_X,\delta_Y)}\\ q_x+2m_x&\leq\frac{\epsilon}{\delta_X}.\end{aligned}$$

So

$$\begin{aligned}\gamma^-(G)&=n-(q_x+2m_x+q_y+2m_y)\\ &\geq\lceil n-(\frac{\epsilon}{\delta_X}+\frac{\epsilon}{1+max(\delta_X,\delta_Y)})\rceil\\ &\geq\lceil n-(\frac{\epsilon}{\delta}+\frac{\epsilon}{1+max(\delta_X,\delta_Y)})\rceil.\end{aligned}$$

In fact, this bound is sharp, it is easy to check that $\gamma^-(K_{1,k})=1=\lceil n-\frac{\epsilon}{\delta}+\frac{\epsilon}{1+max(\delta_X,\delta_Y)}\rceil$.

3 A Lower Bound on k-Subdomination Number of a Graph

The concept of k-subdomination was introduced by Cockayne et al.[2]. In [2], Cockayne et al. established a sharp lower bound on γ_{ks} for trees. Moreover,

they also gave a sharp lower bound on γ_{ks} for trees if $k \leq \frac{n}{2}$ and proposed a conjecture.

Theorem 3 [2]. For any n-vertex tree T and integer $k \in \{1, 2, \ldots, n\}$, $\gamma_{ks} \leq 2(k+1) - n$.

Conjecture 2 [2]. For any n-vertex tree and any k with $\frac{1}{2}n < k \leq n$, $\gamma_{ks} \leq 2k - n$.

In [13], the conjecture was proved and a upper bound for a connected graph was given.

Theorem 4 [13]. For any connected graph of order n and any k with $\frac{1}{2}n < k \leq n$, then

$$\gamma_{ks} \leq 2\lceil \frac{k}{n-k+1} \rceil (n-k+1) - n.$$

In this section, we give a lower bound for a graph G.

Theorem 5. For any graph G of order n and size ϵ,

$$\gamma_{ks} \geq n - \frac{2\epsilon + (n-k)(\Delta+2)}{\delta+1}.$$

Proof. Let f be a k-subdominating function on G with $f(V) = \gamma_{ks}(G)$. Let P and M be the sets of vertices in G that are assigned the values 1 and -1, respectively. Then $|P| + |M| = n$ and $\gamma_{ks}(G) = |P| - |M| = n - 2|M|$. Furthermore, we let

$$\begin{aligned} P_1 &= \{v \in P| \ \ f[v] \geq 1\} \\ P_2 &= P - P_1 \\ M_1 &= \{v \in M| \ \ f[v] \geq 1\} \\ M_2 &= M - M_1 \end{aligned}$$

Clearly, $|P_1| + |M_1| \geq k$. Since each vertex v of P_1 is adjacent to at most $\frac{1}{2}d(v)$ vertices of M, each vertex v of M_1 is adjacent to at most $\frac{d(v)}{2} - 1$ vertices of M. We have

$$\begin{aligned} \delta|M| &\leq \sum_{v \in M} d(v) = \sum_{v \in V} |M \cap N(v)| \\ &\leq \sum_{v \in P_1} \frac{d(v)}{2} + \sum_{v \in M_1} (\frac{d(v)}{2} - 1) + \sum_{v \in P_2 \cup M_2} d(v) \\ &= \frac{1}{2} \sum_{v \in V} d(v) - |M_1| + \frac{1}{2} \sum_{v \in P_2 \cup M_2} d(v) \\ &\leq \epsilon - |M| + \frac{1}{2} \sum_{v \in P_2 \cup M_2} (d(v) + 2) \\ &\leq \epsilon - |M| + (|P_2| + |M_2|) \frac{\Delta + 2}{2}. \end{aligned}$$

As $|P_2| + |M_2| \leq n - k$, it follows that

$$|M| \leq \frac{2\epsilon + (n-k)(\Delta+2)}{2(\delta+1)}$$

Thus

$$\begin{aligned}\gamma_{ks} &= n - 2|M| \\ &\geq n - \frac{2\epsilon + (n-k)(\Delta+2)}{\delta+1}.\end{aligned}$$

This completes the proof of Theorem 5. □

For the graphs in which each vertex has odd degree, the lower bound on γ_{ks} in Theorem 5 can be improved slightly.

Theorem 6. For every graph G in which each vertex vertex has odd degree,

$$\gamma_{ks} \geq n - \frac{2\epsilon + (n-k)(\Delta+2) - k}{\delta+1}.$$

Proof. Let f, P, M, P_1, P_2, M_1 and M_2 be defined as in the proof of Theorem 5.

Since every vertex of G has odd degree, it is easy to see that each vertex v of P_1 is adjacent to at most $\frac{d(v)-1}{2}$ vertices of M, each vertex v of M_1 is adjacent to at most $\frac{d(v)-1}{2} - 1$ vertices of M. Hence we have

$$\begin{aligned}\delta|M| &\leq \sum_{v\in M} d(v) = \sum_{v\in V} |M \cap N(v)| \\ &\leq \sum_{v\in P_1} \frac{d(v)-1}{2} + \sum_{v\in M_1} (\frac{d(v)-1}{2} - 1) + \sum_{v\in P_2\cup M_2} d(v) \\ &= \frac{1}{2}\sum_{v\in V} d(v) - \frac{1}{2}(|P_1| + |M_1|) - |M_1| + \frac{1}{2}\sum_{v\in P_2\cup M_2} d(v) \\ &\leq \epsilon - \frac{1}{2}(|P_1| + |M_1|) - |M| + \sum_{P_2\cup M_2} \frac{d(v)+2}{2} \\ &\leq \epsilon - \frac{1}{2}(|P_1| + |M_1|) - |M| + (|P_2| + |M_2|)\frac{\Delta+2}{2}.\end{aligned}$$

Since $|P_1| + |M_1| \geq k, |P_2| + |M_2| \leq n - k$, we have

$$(\delta+1)|M| \leq \epsilon - \frac{k}{2} + \frac{(\Delta+2)(n-k)}{2}.$$

Hence

$$|M| \leq \frac{2\epsilon + (\Delta+2)(n-k) - k}{2(\delta+1)}.$$

Thus

$$\begin{aligned}\gamma_{ks}(G) &= n - 2|M| \\ &\geq n - \frac{2\epsilon + (\Delta+2)(n-k) - k}{\delta+1}\end{aligned}$$

This completes the proof of Theorem 6. □

By Theorem 5 and Theorem 6, we easily obtain the following lower bounds on γ_{ks} for r-regular graphs.

Theorem 7. Let G be a r-regular graph of order n, then

$$\gamma_{ks} \geq \begin{cases} \frac{r+2}{r+1}k - n, & for\ r\ even \\ \frac{r+3}{r+1}k - n, & for\ r\ odd \end{cases}$$

In the special cases where $k = |V|$ and $k = \lceil \frac{|V|}{2} \rceil$, Theorem 7 deduces to the following results.

Corollary 7 [9]. For every r-regular graph G of order n,

$$\gamma_s(G) \geq \begin{cases} \frac{2n}{r+1}, & for\ r\ odd \\ \frac{n}{r+1}, & for\ r\ even \end{cases}$$

and the bounds are sharp.

Corollary 8 [9]. For every r-regular($r \geq 2$) graph G of order n,

$$\gamma_{maj}(G) \geq \begin{cases} \frac{1-r}{2(r+1)}n, & for\ r\ odd \\ \frac{-r}{2(r+1)}n, & for\ r\ even \end{cases}$$

and the bounds are sharp.

References

1. I. Broere, J.H. Hattingh, M.A. Henning and A.A. McRae, Majority domination in graphs, Discrete Math. 138(1995) 125-135.
2. E.J. Cockayne and C.M. Mynhardt, On a generalization of signed dominating function of graphs, Ars Combin. 43(1996) 235-245.
3. J.E. Dunbar, S.T. Hedetniemi, M.A. Henning, A. McRae, Minus domination in graphs, Discrete Math. 199(1999) 35-47.
4. J.E. Dunbar, W. Goddard, S.T. Hedetniemi, M.A. Henning, A. McRae, The algorithm complexity of minus domination in graphs, Discrete Appl. Math. 68(1996) 73-84.
5. J.E. Dunbar, S.T. Hedetniemi, M.A. Henning and P.J. Slater, Signed domination in graphs, eds. Y. Alavi and A. Schwenk. Graph Theory, Combinatorics, and Applications, New York: Wiley, 1995, 311-321.
6. O. Favaron, Signed domination in regular graphs, Discrete Math. 158(1996) 287-293.
7. M.A. Henning and P.J. Slater, Inequality relation domination parameters in cube graphs, Discrete Math. 158(1996) 87-98.
8. M.A. Henning and H.R. Hind, Strict Majority Functions on Graphs, Journal of Graph Theorem 28(1998) 49-56.
9. M.A. Henning, Domination in regular graphs, Ars Combinatoria 43(1996) 263-271.
10. S.T. Hedetniemi, P.J. Slater, Fundamenters of Domination in Graphs, Marcel Dekker, New York, 1998.

11. T.W. Haynes, S.T. Hedetniemi, P.J. Slater(Eds.) Domination in Graphs: Advanced Topics, Marcel Dekker, New York, 1998.
12. Kang Liying, Shan Erfang, Lower bounds on dominating functions in graphs, Ars Combin. 56(2000), 121-128.
13. Kang Liying, Dang Chuangyin, Cai Maocheng, Shan Erfang, Upper bounds for k-subdomination number of graphs, Discrete Math. (to appear)

Edge Connectivity vs Vertex Connectivity in Chordal Graphs

L. Sunil Chandran*

Indian Institute Of Science, Bangalore, India
sunil@csa.iisc.ernet.in

Abstract. It is well known that in a graph, $\kappa(G) \leq \lambda(G) \leq \delta(G)$, where $\kappa(G), \lambda(G)$ and $\delta(G)$ denote the *vertex connectivity, edge connectivity* and the *minimum degree* of G, respectively. We show that in chordal graphs, if $\lambda(G) \neq \delta(G)$, then $\lambda(G) \geq 2\kappa(G) - 1$. In contrast, in a general graph $\lambda(G)$ can be equal to $\kappa(G)$.

1 Introduction

1.1 Chordal Graphs: Some Definitions and Notation

Let C be a cycle in a graph G. A chord of C is an edge of G joining two vertices of C which are not consecutive. A graph G is called a chordal (or triangulated) graph iff every cycle in G of length 4 or more has a chord. Chordal graphs arise in many applications (see [3,4,5]). Chordal graphs constitute one of the most important subclasses of perfect graphs [3].

Let $G = (V, E)$ be a connected undirected graph. Throughout this paper we will use V for the set of vertices of G and E for the set of edges. $|V|$ will be denoted by n. $N(v)$ will denote the set of neighbours of v, that is $N(v) = \{u \in V : (u, v) \in E\}$. For $A \subseteq V$, we use $N(A)$ to denote the set $\bigcup_{v \in A} N(v) - A$. The subgraph of G induced by the nodes in A will be denoted by $G(A)$.

A bijection $f : V \to \{1, 2, \cdots, n\}$ is called an ordering of the vertices of G. Then $f(v)$ is referred to as the number associated with the vertex v, or simply the *number of* v with respect to the ordering f. Given an ordering f of a graph G, we define the following terms.

Definition 1. *Let $A \subseteq V$. The highest(A) is defined to be the vertex with the highest number in A.*

Definition 2. *A path $P = (w_1, w_2, \cdots, w_k)$ in G is called an increasing path, iff $f(w_1) < f(w_2) < \cdots < f(w_k)$. It is called a decreasing path iff $f(w_1) > f(w_2) > \cdots > f(w_k)$. A single node can be considered as either increasing or decreasing.*

* Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, 560012. sunil@csa.iisc.ernet.in

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 384–389, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Definition 3. *A vertex $u \in N(v)$ is called a higher neighbour of v iff $f(u) > f(v)$. The set of higher neighbours of v will be denoted by $N_h(v)$, ie*

$$N_h(v) = \{u \in N(v) : f(u) > f(v)\}$$

Definition 4. *An ordering f of G is called a perfect elimination ordering (PEO) iff for each $v \in V$, $G(\{v\} \bigcup N_h(v))$ is a complete subgraph (clique) of G. Then $f(v)$ will be called the PEO number of v.*

Note that every graph may not have a PEO. The following theorem is well known (see [3]).

Theorem 1. *An undirected graph G is chordal if and only if there exists a PEO for G.*

Note that there can be more than one PEO for a given chordal graph. We just use $PEO(v)$ to denote the PEO number of v, since our arguments are valid with respect to any PEO. Whenever we talk of a chordal graph, we will assume a PEO for it and the words *highest, increasing path* etc will be used with the respect to this PEO.

1.2 Edge Connectivity and Vertex Connectivity

The *vertex connectivity* $\kappa(G)$ of an undirected graph is defined to be the minimum number of vertices whose removal results in a disconnected graph or a trivial graph (ie, single node). A subset $S \subset V$ is called a *separator* if $G(V - S)$ has at least two components. Then, $\kappa(G)$ is the size of the minimum sized separator. Note that a complete graph has no separator and its (vertex) connectivity is by definition $n - 1$.

The following result called *Menger's theorem* is well known among graph theorists. An elegant proof due to Dirac is given in [2].

Theorem 2. Menger's theorem : *The minimum number of vertices separating two nonadjacent nodes s and t is equal to the maximum number of vertex disjoint $s - t$ paths.*

Thus, there are at least $\kappa(G)$ vertex disjoint paths between every pair of nonadjacent nodes of G.

The *edge connectivity* $\lambda(G)$ of a graph G is defined to be the minimum number of edges whose removal results in a disconnected graph or a trivial graph. A *minimum cut* is a cut with its size $= \lambda(G)$. Note that the minimum degree $\delta(G)$ of the graph is an upper bound for mincut size (ie edge connectivity).

1.3 Our Result

It is well known that the following inequality (due to Whitney [6]) holds in a general undirected graph.

$$\kappa(G) \leq \lambda(G) \leq \delta(G)$$

where $\delta(G)$ is the minimum degree of the graph. For a discussion of this inequality and for related work see the chapter on connectivity in [2]. In [1], Chartrand and Harary show that for all integers a, b, c, such that $0 < a \leq b \leq c$, there exists a graph G with $\kappa(G) = a$, $\lambda(G) = b$ and $\delta(G) = c$.

In this paper, we study this inequality when restricted to the class of chordal graphs. We show that there is a "gap" between $\lambda(G)$ and $\kappa(G)$ in chordal graphs provided $\lambda(G) \neq \delta(G)$. More specifically, if $\lambda(G) \neq \delta(G)$, then $\lambda(G) \geq 2\kappa(G) - 1$.

2 Edge Connectivity vs Vertex Connectiviy in Chordal Graphs

First we make some simple observations about mincuts in undirected graphs. Remember that mincut is a minimum collection of edges which when removed disconnects the graph. Let V be partitioned to nonempty disjoint sets A and B by a cut. We denote the set of edges in the cut by (A, B).

Lemma 1. *Let (A, B) be a mincut of a connected undirected graph G. Then $G(A)$ and $G(B)$ are connected.*

Proof : Suppose that $G(A)$ is not connected. Let $G(A_1)$ be a connected component of $G(A)$, where $A_1 \subset A$. Let $A_2 = A - A_1$. Clearly $(A_1, B \bigcup A_2)$ is a cut of G and $|(A_1, B \bigcup A_2)| < |(A, B)|$, since $(A_1, B \bigcup A_2) \subset (A, B)$. But this is a contradiction since (A, B) is assumed to be the mincut. □

Lemma 2. *Let (A, B) be a mincut of G, and let $\lambda(G) = |(A, B)| < \delta(G)$. Then there exists a node $x \in A$, such that $x \notin N(B)$. Similarly $\exists y \in B$ such that $y \notin N(A)$.*

Proof : Suppose that every node in A is adjacent to some node in B. Let u be a node in A. Let $F = (A, B)$, be the mininum cut. We have

$$d(u) = |N(u) \bigcap B| + |N(u) \bigcap A| \leq |N(u) \bigcap B| + |A| - 1$$

But

$$|F| = \sum_{x \in A} |N(x) \bigcap B| \geq |N(u) \bigcap B| + |A| - 1$$

since each term in the sum should be at least 1 by the assumption that every node in A is adjacent to at least one node in B. It follows that $d(u) \leq |F|$. But by assumption, $d(u) \geq \delta(G) > |F|$. Thus we have a contradiction and we conclude that there is a node $x \in A$ such that $x \notin N(B)$. By similar arguments, $\exists y \in B$ such that $y \notin N(A)$. □

Corollary 1. *If (A, B) is a mincut of G, and if $\lambda(G) \neq \delta(G)$ then $|A| > \kappa(G)$, $|B| > \kappa(G)$.*

Proof : By lemma 2, there exists a node $x \in A$ such that $x \notin N(B)$. Let $y \in B$. It is clear that $N(B)$ separates x from y. Thus $|N(B)| \geq \kappa(G)$. But $N(B) \subseteq A - \{x\}$. It follows that $\kappa(G) < |A|$. Similarly $\kappa(G) < |B|$. □

A chordless path from u to v is defined to be a path from u to v in G with no two non consecutive nodes of the path being adjacent. The reader can easily verify that if there is a path between u and v then there is a chordless path also. For example, the shortest path between u and v has to be a chordless path.

Lemma 3. *Let $P = (w_1, w_2, \cdots, w_k)$ be a chordless path in a chordal graph G and let $w_i = highest(P)$. Then $(w_1, w_2, \cdots, w_i)$ is an increasing path while $(w_i, w_{i+1}, \cdots, w_k)$ is a decreasing path.*

Proof: Consider first the path $(w_1, w_2, \cdots, w_i)$. Since $w_i = highest(P)$, this cannot be a decreasing path. If $i \leq 2$, obviously this is an increasing path. Let $i > 2$. If $(w_1, w_2, \cdots, w_i)$ is not an increasing path, then there should be a node $w_j \in P$, $1 < j < i$, such that $PEO(w_j) < PEO(w_{j+1})$ and $PEO(w_j) < PEO(w_{j-1})$. Then $\{w_{j-1}, w_{j+1}\} \subseteq N_h(w_j)$. By definition of PEO, $N_h(w_j)$ forms a clique, and we have $(w_{j-1}, w_{j+1}) \in E$. So (w_{j-1}, w_{j+1}) forms a chord for P, contradicting the chordless property of P. We infer that $(w_1, w_2, \cdots, w_i)$ is an increasing path. A similar argument shows that $(w_i, w_{i+1}, \cdots, w_k)$ should be a decreasing path. □

Corollary 2. *Let $P = (w_1, w_2, \cdots, w_k)$ be a chordless path and $w_k = highest(P)$. Then P is an increasing path.*

Corollary 3. *Let $P = (w_1, w_2, \cdots, w_k)$ be a chordless path and $PEO(w_1) < PEO(w_k)$. Then $PEO(w_2) > PEO(w_1)$.*

Let $G = (V, E)$ be a chordal graph and (A, B) be a mincut of G. Without loss of generality assume that B contains the node with PEO number n. Let $x = highest(A)$. Let $B_1 = B \bigcap N(x)$ and $B_2 = B - B_1$.

Lemma 4. *If $|B_1| < \kappa(G)$, and $\lambda(G) \neq \delta(G)$, then $PEO(highest(B_2)) < PEO(highest(A))$.*

Proof : Let $x = highest(A)$ and $y = highest(B_2)$. (Note that since $|B| > \kappa(G)$ (by corollary 1), and $|B_1| < \kappa(G)$, B_2 is nonempty and therefore $highest(B_2)$ exists). First observe that $N_h(x) \subseteq B_1$, since every neighbour of x with a higher PEO number has to be in B. We will prove the lemma by contradiction. In other words, we will show that if $PEO(y) > PEO(x)$, then $|N_h(x)| \geq \kappa(G)$, thereby contradicting the assumption that $|B_1| < \kappa(G)$, since $N_h(x) \subseteq B_1$. Note that y is non adjacent to x since $y \in B_2$. We will show that $N_h(x)$ separates x from y. Consider any path P from x to y. Clearly there is a chordless path P' from x to y, containing only the nodes in P. Let w be the second node in P', starting from x. By corollary 3 $PEO(w) > PEO(x)$. Thus $w \in N_h(x)$. We have shown that in every path P from x to y there is a node which is in $N_h(x)$. Therefore $N_h(x)$ is a separator and $|N_h(x)| \geq \kappa(G)$. This in turn implies $|B_1| \geq \kappa(G)$, contradicting the assumption. □

Theorem 3. *For a chordal graph G, if $\lambda(G) \neq \delta(G)$, then $\lambda(G) \geq 2\kappa(G) - 1$.*

Proof : Let (A, B) be a mincut of G. Remember that $|(A, B)| = \lambda(G) < \delta(G)$. Without loss of generality, let B contain the node with PEO number n. Let $x = highest(A)$. By lemma 2, there is a node $a \in A$, and a node $b \in B$ such that $a \notin N(B)$ and $b \notin N(A)$. We observe first that $a \neq x$. To see this, consider a chordless path from x to the node with PEO number n, which is in B by assumption. By corollary 2, this is an increasing path and since $x = highest(A)$, the second node in this path after x has to be in B. It follows that $x \in N(B)$. Therefore $x \neq a$.

Now since a and b are nonadjacent, by Menger's theorem (Theorem 2), there exist at least $\kappa(G)$ vertex disjoint paths between them. One of these paths may contain x. Consider $\kappa(G) - 1$ vertex disjoint paths between a and b, which do not contain x. Each of these paths should exit A, thus contributing at least one edge to the mincut (A, B). Collect one such edge from each path and let F_1 be the set of $\kappa(G)-1$ edges collected this way. We emphasize that none of the edges in F_1 is incident on x, a or b. (Since we have avoided the path containing the node x; and none of the edges incident on a or b are in (A, B), by the choice of a and b). Moreover no two edges in F_1 will share a common end point since they are all coming from distinct vertex disjoint paths.

Now let $B_1 = N(x) \bigcap B$, be the set of neighbours of x which are in B. Let $F_2 = \{(x, u) : u \in B_1\}$, the set of edges starting from x and ending at a node in B_1. Clearly $F_2 \subset (A, B)$ and $F_2 \bigcap F_1 = \phi$. If $|B_1| \geq \kappa(G)$, $|F_2| \geq \kappa(G)$ also and we have $\lambda(G) = |(A, B)| \geq |F_1| + |F_2| \geq 2\kappa(G) - 1$, as required by the theorem. If $|B_1| < \kappa(G)$, let $B_2 = B - B_1$. B_2 is nonempty since $|B| > \kappa(G)$ by corollary 1. Let $y = highest(B_2)$. By lemma 4, $PEO(y) < PEO(x)$. Noting that x and y are nonadjacent, there exist $\kappa(G)$ vertex disjoint paths between them, by Menger's theorem (Theorem 2). Clearly at least $l = \kappa(G) - |B_1|$ of them should be paths which consist of vertices in $A \bigcup B_2$ only. Let these paths be $P_1, P_2, \cdots, P_l$. For each path P_i, $1 \leq i \leq l$, there exists a chordless path P_i' from y to x which uses only the nodes of P_i. Since x and y are nonadjacent, each of these chordless paths should contain at least 3 nodes. Let u_i be the second node in P_i', starting from y. We claim that $u_i \in A$. This is because $PEO(u_i) > PEO(y)$ by corollary 3 and $y = highest(B_2)$. Since $u_i \notin B_1$, it has to be in A. Thus $(u_i, y) \in (A, B)$. Now consider a chordless path in B from y to the node having PEO number n. Since $G(B)$ is connected by lemma 1, such a chordless path exists and clearly it is an increasing path. Let w be the second node in this path starting from y. Clearly u_i and w are neighbours of y with higher PEO number than that of y itself, ie $\{w, u_i\} \subseteq N_h(y)$. By definition of PEO, $N_h(y)$ forms a clique and therefore $(u_i, w) \in E$. Clearly since $w \in B$ and $u_i \in A$, $(u_i, w) \in (A, B)$. Since no two edges in F_1 shares a common end point, one of the two edges (u_i, y) or (u_i, w) is not in F_1. Now for each P_i', $1 \leq i \leq l$, add to F_3, one of $(u_i, w), (u_i, y)$ which ever is not in F_1. Clearly $|F_3| = l = \kappa(G) - |B_1|$. Also $F_3 \bigcap F_1 = \phi$ and $F_3 \bigcap F_2 = \phi$. Since $F_1 \bigcup F_2 \bigcup F_3 \subseteq (A, B)$, we have $\lambda(G) = |(A, B)| \geq |F_1| + |F_2| + |F_3| = 2\kappa(G) - 1$, as required by the theorem. This completes the proof. □

References

1. G. Chartrand, F.Harary, *Graphs with prescribed connectivities* Theory of Graphs (P.Erdos and G.Katona, eds.) Akademiai Kiado, Budapest, 1968, 61-63.
2. F.Harary, *Graph Theory.* Addison-Wesley Reading, MA, 1969.
3. M.C.Golumbic, *Algorithmic Graph Theory And Perfect Graphs.* Academic Press, New York, 1980.
4. R.H.Mohring, *Graph Problems Related To Gate Matrix Layout And PLA Folding.* Computational Graph Theory, Pages 17-52, Springer, Wein, New York,1990.
5. M.Yannakakis, *Computing The Minimum Fill-in Is NP-Complete* SIAM J. on Alge. Discre. Math. 2 77-79, 1981.
6. H.Whitney, *Congruent Graphs and the connectivity of graphs.* American J. Math, 54, 1932, 150-168

Changing the Diameter of Graph Products

Ting-Yi Sung[1] and Jeng-Jung Wang[2]

[1] Institute of Information Science,
Academia Sinica, Taipei, Taiwan 115, R.O.C.
tsung@iis.sinica.edu.tw
[2] Department of Information Engineering,
I-Shou University, Kaohsiung, Taiwan 840, R.O.C.
jjwang@isu.edu.tw

Abstract. Graham and Harary [3]studied how the diameter of hypercubes can be affected by increasing and decreasing edges. Since many networks are constructed by graph products, e.g., tori and meshes, in this paper we study how the diameter of graph products, particularly Cartesian product and Kronecker product, can be changed by adding and removal of edges. We study Cartesian products on paths, cycles, trees, and hypercubes. The diameter of the Kronecker product of two graphs is in general difficult to find. We in particular study the Kronecker product of two cycles.

1 Introduction

Let $G = (V, E)$ denote a graph. For any two vertices u, v in G, let $d_G(u, v)$ denote the *distance* between u and v which is defined as the length of a shortest path between them. The *diameter* of G, denoted by $D(G)$, is defined as $D(G) = \max_{u,v \in V} d_G(u, v)$. Graham and Harary [3] studied how the diameter of hypercubes can be affected by increasing and decreasing edges. We are interested in the following measures defined in [3]:

$D^-(G)$**:** the least number of edges whose addition to G decreases the diameter;
$D^+(G)$**:** the least number of edges whose deletion from G increases the diameter.

For example, since deleting an edge from a cycle of length m, denoted by C_m, increases the diameter from $\lfloor m/2 \rfloor$ to $m - 1$, it follows that $D^+(C_m) = 1$. The superscript of D indicates the "change" of diameter, i.e., "-" to mean decreasing the diameter, and "+" increasing the diameter.

Many popular networks are constructed by graph products. We are particularly interested in Cartesian product and Kronecker product. Let G_1 and G_2 be two graphs, and we write $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ throughout the paper. The *Cartesian product* $G = (V, E)$ of G_1 and G_2, denoted by $G = G_1 \cdot G_2$, is given by $V = V_1 \times V_2$, and

$$E = \{(u_1u_2, v_1v_2) \mid u_1 = v_1 \text{ and } (u_2, v_2) \in E_2, \text{ or } u_2 = v_2 \text{ and } (u_1, v_1) \in E_1\}.$$

The *Kronecker product* $G = (V, E)$, also known as *direct product* and *tensor product*, of G_1 and G_2, denoted by $G = G_1 \times G_2$, is given by $V = V_1 \times V_2$ and

$$E = \{(u_1u_2, v_1v_2) \mid (u_1, v_1) \in E_1 \text{ and } (u_2, v_2) \in E_2\}.$$

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 390–394, 2001.
© Springer-Verlag Berlin Heidelberg 2001

A torus is the Cartesian product of two cycles, and mesh the Cartesian product of two paths. The diagonal toroidal mesh discussed in [6] is the Kronecker product of two cycles. In [2], the authors studied the Cartesian products of complete binary trees, shuffle-exchange graphs and de Bruijn graphs, which possess various good properties.

We adopt the following notation. Let P_n and C_n denote a path and a cycle, respectively, of n vertices labeled by $0, 1, \ldots, n-1$. Let Q_n be a hypercube of dimension n of 2^n vertices with each vertex denoted by an n-bit binary representation, where two vertices are connected by an edge if and only if two vertices differ at exactly one bit.

Since we work on graph products, for ease of exposition we use (), [] and $\langle\rangle$ to delimit a vertex, a pair of vertices and an edge, respectively. For example, each vertex in $G = G_1 \cdot G_2$ is denoted by (v_1, v_2), where $v_1 \in V_1$ and $v_2 \in V_2$. An edge in $G_1 \cdot G_2$ is denoted by $\langle (u_1, u_2), (v_1, v_2) \rangle$. We also use $\langle$ and $\rangle$ to delimit a path.

2 Preliminaries

Let $\delta(G)$ denote the minimum degree of G, i.e., $\delta(G) = min\{\delta_G(v) \mid v \text{ in } G\}$, where $\delta_G(v)$ denotes the degree of a vertex v in G. It is obvious that $D^+(G) \leq \delta(G)$.

It is known that $D(P_n) = n-1$, $D^-(P_n) = 1$ and $D^+(P_n) = 1$. The diameter of cycle C_n is given by $D(C_n) = \lfloor n/2 \rfloor$. Obviously, $D^+(C_n) = 1$. It is shown in [4] that $D^-(C_n) = 2$ for $n \geq 8$ by adding the following two edges:

$$e_1 = \langle 0, \lfloor n/2 \rfloor \rangle,$$
$$e_2 = \begin{cases} \langle \lfloor n/4 \rfloor, \lceil 3n/4 \rceil \rangle & \text{for } n = 2 \bmod 4, \\ \langle \lfloor n/4 \rfloor, \lfloor 3n/4 \rfloor \rangle & \text{for } n = 0, 1, 3 \bmod 4. \end{cases} \quad (1)$$

Hypercubes have been extensively studied. Let v be a vertex in Q_n. It is known that Q_n is n-connected. Let x and y be two vertices in Q_n. We use $h[x, y]$ to denote the *hamming distance* between x and y which is defined as the number of differing bits in x and y. It is shown in [3] that $D^-(Q_n) = 2$ by adding two diagonal edges in any subcube Q_2 of Q_n, e.g., $\langle v_0, 00\ldots011 \rangle$ and $\langle 00\ldots001, 00\ldots010 \rangle$. It is well known and shown in [5] that there are n internally node-disjoint paths between x and y, $h[x, y]$ of them having length of $h[x, y]$ and $n - h[x, y]$ having length of $h[x, y] + 2$. It follows that $D^+(Q_n) = n-1$ by deleting $n-1$ edges in a subcube Q_{n-1} of Q_n which are incident at any specific vertex [3]. For example, deleting $\langle 000, 001 \rangle$ and $\langle 000, 010 \rangle$ from Q_3, the distance from 000 to 011 becomes 4, so is the diameter.

3 Cartesian Products

Cartesian products also preserve the node symmetry property of its two primitive graphs. To be specific, let G_1, G_2 be node-symmetric. The following lemma can be easily proved.

Lemma 1 *Let G_1, G_2 be node-symmetric. Then $G_1 \cdot G_2$ is also node-symmetric.*

The following lemma can be easily shown.

Lemma 2 $D(G_1 \cdot G_2) = D(G_1) + D(G_2)$.

For example, $D(T_{m,n}) = D(C_m) + D(C_n) = \lfloor m/2 \rfloor + \lfloor n/2 \rfloor$. Furthermore, it can be easily verified that

$$D^+(G_1 \cdot G_2) \geq min\{D^+(G_1), D^+(G_2)\} + 1 \quad (2)$$

$$D^-(G_1 \cdot G_2) \geq min\{D^-(G_1), D^-(G_2)\}. \quad (3)$$

One may ask whether (2) and (3) always hold as inequalities. To answer this, consider $P_m \cdot P_n$ which is called mesh of size m by n. It can be easily verified that $D^-(P_m \cdot P_n) = 1$ and $D^+(P_m \cdot P_n) = 2$, i.e., (2) and (3) hold as equalities. We subsequently ask how to characterize those G_1 and G_2 for (2) and (3) to hold, respectively, as equalities. Another interesting question is whether we can tighten the lower bound in (2).

On the other hand, we wonder whether (2) and (3) can hold as equalities when G_1 and G_2 are node-symmetric. Consider $T_{m,n} = C_m \cdot C_n$, where C_i is node-symmetric. We can show the following theorem with proof omitted.

Theorem 1 $D^+(C_m \cdot C_n) = \begin{cases} 2 & \text{for } m, n \text{ odd,} \\ 3 & \text{for one of } m, n \text{ odd,} \\ 4 & \text{for } m, n \text{ even.} \end{cases}$

The above theorem illustrates that (2) does not necessarily hold as equality when G_1 and G_2 are node-symmetric. It is shown in [4] that $D^-(C_n) = 2$ for $n \geq 8$ and $D^-(T_{m,n}) = 2$ for $m \geq 12$ or $n \geq 12$, and (3) holds as equality.

In what follows, we study $D^+(G_1 \cdot G_2)$ and $D^-(G_1 \cdot G_2)$ for G_1, G_2 to be paths, cycles, hypercubes and trees.

Lemma 3 *(i)* $D^-(C_m \cdot P_n) = 1$ *for all* $n \geq 4$.
(ii) $D^+(C_m \cdot P_n) = \begin{cases} 2 & \text{for } m \text{ odd,} \\ 3 & \text{for } m \text{ even.} \end{cases}$
(iii) $D^-(Q_m \cdot P_n) = 1$ *for* $n \geq 3$.
(iv) $D^+(Q_m \cdot P_n) = m$.
(v) $D^-(Q_m \cdot C_n) = 2$ *for* $n \geq 8$.
(vi) $D^+(Q_m \cdot C_n) = \begin{cases} m+2 & \text{for } n \text{ even,} \\ m+1 & \text{for } n \text{ odd.} \end{cases}$.
(vii) $D^-(T_1 \cdot T_2) = \min\{D^-(T_1), D^-(T_2)\}$ *and* $D^+(T_1 \cdot T_2) = 2$, *where* T_1 *and* T_2 *are arbitrary trees.*

After careful observation of the above theorem, we have the following conjecture.

Conjecture 1 *If G_1 and G_2 are node-symmetric, then* $D^-(G_1 \cdot G_2) = \min\{D^-(G_1), D^-(G_2)\}$.

Furthermore, we conjecture that the lower bound in (2) can be increased as stated below.

Conjecture 2 $D^+(G_1 \cdot G_2) \geq \max\{D^+(G_1), D^+(G_2)\} + 1$.

4 Kronecker Product of Cycles

Kronecker product preserves good properties of its two primitive graphs. For example, if G_1 and G_2 are node-symmetric, $G_1 \times G_2$ is also node-symmetric.

The diameter of a Kronecker product on two arbitrary graphs is in general difficult to find. In this section, we consider Kronecker products of two cycles C_m and C_n. In this section, we calculate $D(C_m \times C_n)$ using different approach from that in [6] and find $D^+(C_m \times C_n)$ and give an upper bound for $D^-(C_m \times C_n)$. If both m and n are even, $C_m \times C_n$ becomes disconnected. In this section, we assume that at least one of m and n is odd.

In C_k, let $N(i)$ denote the ith-neighborhood of vertex 0 which is defined as the set of the vertices in C_k that can be reached from 0 at the ith step. Then

$$N(i) = \begin{cases} \{1, 3, \ldots, i, k-i, k-i+2, \ldots, k-1\} & \text{for } i \text{ odd}, \\ \{0, 2, 4, \ldots, i, k-i, k-i+2, \ldots, k-2\} & \text{for } i \text{ even}. \end{cases}$$

For example, given $k = 11$, we have

$$N(1) = \{1, 10\},\ N(2) = \{0, 2, 9\},\ N(5) = \{1, 3, 5, 6, 8, 10\},$$
$$N(6) = \{0, 2, 4, 5, 6, 7, 9\},\ N(9) = \{1, 2, 3, 4, \ldots, 10\}.$$

With careful observation, we can obtain the following remark.

Remark 1 *(i)* $i = \lfloor k/2 \rfloor$ *is the smallest integer satisfying* $\bigcup_{j=1}^{i} N(j) = \{0, 1, 2, \cdots, k-1\}$, *and in particular,* $N(\lfloor k/2 \rfloor - 1) \cup N(\lfloor k/2 \rfloor) = \{0, 1, 2, \ldots, k-1\}$.
(ii) $\lfloor k/2 \rfloor \notin N(i)$ *for all* $i \leq \lfloor k/2 \rfloor - 1$.
(iii) $N(i) \subseteq N(i+2)$ *for* $1 \leq i \leq n-3$, *and in particular,* $N(i) = N(i+2)$ *for* k *even and* $k/2 \leq i \leq k-3$.

In this section, let $G = C_m \times C_n$ and $d^* = D(G)$. Since G is node-symmetric, we can consider d^* as the longest distance from $(0, 0)$ to all of the other vertices. In order to distinguish the ith-neighborhood of vertex 0 in C_m and C_n, we use $N_1(i)$ and $N_2(i)$ to denote that in C_m and C_n, respectively. Henceforth, we use $N(i)$ to denote the ith-neighborhood of vertex $(0, 0)$ in G without ambiguity, i.e., the set of vertices in G that can be reached from $(0, 0)$ at the ith step. It follows that $N(i) = N_1(i) \times N_2(i)$. The diameter d^* is the smallest integer i satisfying $\bigcup_{j=1}^{i} N(j) = V_1 \times V_2$. It can be easily verified that $d^* \geq max\{\lfloor m/2 \rfloor, \lfloor n/2 \rfloor\}$.

In this paper we consider that both m and n are odd and assume without loss of generality that $n \geq m$. The case with exactly one of m and n being odd can be similarly treated. We have the following results.

Lemma 4 *Let both* m *and* n *be odd, and* $n \geq m$. *Then*

$$D(C_m \times C_n) = \begin{cases} m-1 & \text{for } m = n, \\ m & \text{for } m+2 \leq n \leq 2m+1, \\ \lfloor n/2 \rfloor & \text{for } n \geq 2m+3. \end{cases}$$

Proof: We assume without loss of generality that $m \leq n$. To find d^*, we distinguish the following cases:

Case (i): $m = n$.

Since $N_1(m-1) = V_1$ and $N_1(i) = N_2(i)$, it follows that $N(m-1) = V_1 \times V_2$ and $d^* \leq m-1$. On the other hand, since $N_1(m-1) - N_1(m-2) = \{0, m-1\}$ and $N_1(m-2) - N_1(m-3) = \{1\}$, it follows that the vertices $(0,1)$, $(1,0)$, $(0,m-1)$, and $(m-1,0)$ are not in $N(m-2)$. Therefore, $m-2 < d^* \leq m-1$, i.e., $d^* = m-1$.

Case (ii): $m+2 \leq n \leq 2m+1$.

Since $m \geq \lfloor n/2 \rfloor$, it follows that $N_2(m-1) \cup N_2(m) = V_2$. Furthermore, since $N_1(m) = V_1$, it follows that $N(m) = V_1 \times V_2$ and thus $d^* \leq m$. On the other hand, since $N_1(m-1) - N_1(m-2) = \{0\}$, $N(m-1)$ does not contain the vertex set $\{(0,j) \mid j \in N_2(m) - N_2(m-1)\}$. It follows that $m-1 < d^* \leq m$, i.e., $d^* = m$.

Case (iii): $n \geq 2m+3$, i.e., $\lfloor n/2 \rfloor \geq m+1$.

Note that $N_1(\lfloor n/2 \rfloor - 1) = N_1(\lfloor n/2 \rfloor) = V_1$ and $d^* \geq \lfloor n/2 \rfloor$. On the other hand, since $N_2(\lfloor n/2 \rfloor - 1) \cup N_2(\lfloor n/2 \rfloor) = V_2$ as stated in Remark 1(i), it follows that $\bigcup_{i=1}^{\lfloor n/2 \rfloor} N_1(i) \times N_2(i) = V$. Thus $d^* = \lfloor n/2 \rfloor$. The lemma follows. □

Our proof method for Lemma 4 is different from that in [6]. We then can show the following theorem with proof omitted.

Theorem 2 *Let m, n be odd, and $n \geq m$.*

$$D^+(C_m \times C_n) = \begin{cases} 1 & \text{if } n = 2m+1, \\ 2 & \text{otherwise.} \end{cases}$$

We can also give a construction of adding edges such that the diameter is reduced which also provides an upper bound of $D^-(C_m \times C_n)$.

References

1. J.-C. Bermond and B. Bollobas, The diameter of graphs – A survey, *Congr. Numer.* 32, pp. 3–27, 1981.
2. K. Efe and A. Fernández, Products of networks with logarithmic diameter and fixed degree, *IEEE Trans. Parallel and Distributed Systems* 6, pp. 963–975, 1995.
3. N. Graham and F. Harary, Changing and unchanging the diameter of a hypercube, *Discrete Applied Mathematics* 37/38, pp. 265–274, 1992.
   MA: Addison-Wesley, 1969.
4. T.Y. Sung, T.Y. Ho and D. Ferrero, Diameter variability of hypercubes and cyclic networks, manuscript, 2000.
5. Y. Saad and M.H. Schultz, Topological properties of hypercubes, *IEEE Trans. Computers* 37, pp. 867–872, 1988.
6. K.W. Tang and S.A. Padubidri, Diagonal and toroidal mesh networks, *IEEE Trans. Computers* 43, pp. 815–826, 1994.

Plane Graphs with Acyclic Complex*

Baogang Xu

Institute of System Sciences, Academy of Math. & System Sciences, CAS,
Zhongguancun, Beijing, 100080, P.R. China,
bgxu@staff.iss.ac.cn

Abstract. All vertex-sets of cliques of G form the simplexes of an abstract complex $C(G)$, which is called the *clique complex* of G. It is proved that the clique complex of a plane graph G is acyclic if and only if G is contractible.

1 Introduction

Graphs considered are finite and simple, complexes considered are finite. Undefined signs can be found in [1,2]. An n-complex is a complex of dimension n. n-simplex, n-chain and n-cycle are defined analogously. The topology properties of graphs have important applications in image analyses[3].

Given an n-complex K and integer $0 \leq q \leq n$, $H_q(K)$ and $\chi(K)$ denote the q_{th} homology group of K with coefficients integers J and the *Euler characteristic* of K, respectively. ∂ is the *boundary operator* on K. An n-complex K is called acyclic if $H_q(K) = 0$ for $q > 0$. Two complexes K and L are homologous, denoted by $K \sim L$, if $H_q(K) = H_q(L)$ for each $q \geq 0$.

Given a graph $G = (V(G), E(G))$. $N_G(u)$ denotes the set of vertices adjacent to u in G. $d_G(u)$ and $\delta(G)$ denote the degree of u and the minimum degree of G, respectively. A k-vertex is vertex of degree k. For a set $S \subset V(G)$, $G[S]$ denotes the subgraph of G induced by S. A *clique* is a complete subgraph of G. All vertex-sets of cliques of G form the simplexes of an abstract complex $C(G)$, which is called the *clique complex* of G. For convenient, we use $H_q(G)$ to denote the q_{th} homology group of $C(G)$, and use a sequence of vertices of G to denote an oriented simplex of $C(G)$.

Defination 11 [4] *(Ivashchenko) A family $\Re$ of graphs is called contractible if $K_1 \in \Re$ and each graph in $\Re$ can be obtained from K_1 by following contractible transformations: (1) Deleting a vertex v from G, where $G[N_G(v)] \in \Re$; (2) Adding a vertex $v \notin V(G)$ to G and joining it to each vertex of an induced subgraph $G_1 \in \Re$ of G; (3) Deleting an edge $uv \in E(G)$, where $G[N_G(u) \bigcap N_G(v)] \in \Re$; (4) Adding an edge $uv \notin E(G)$ to G, where $G[N_G(u) \bigcap N_G(v)] \in \Re$. All graphs in $\Re$ are called contractible.*

Theorem 11 [4] *(Ivashchenko) Contractible transformations do not change the homology groups of graphs. Especially, if a graph G is contractible, then $H_0(C(G)) = J$, and $H_q(C(G)) = 0, q \geq 1$.*

* Research partially supported by the Innovation Fund. of CAS.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 395–399, 2001.
© Springer-Verlag Berlin Heidelberg 2001

By Theorem 11, $C(G)$ is acyclic while G is contractible. We guess that each connected graph with acyclic clique complex is contractible. Our main result is

Theorem 12 *Let G be a connected plane graph. Then $C(G)$ is acyclic iff G is contractible.*

2 Homology Groups of Plane Graphs

Let G be a plane graph. When we say that H is a subgraph of G, it means that the plane embedding of H is a restriction of G on H. For a face f of G, we use $b(f)$ to denote a sequence of vertices in clockwise order on the boundary of f. A face f of G is called a k-face if f is incident with exactly k edges. A *maximal subgraph* of G is a maximal plane graph which is a subgraph of G. Let C be a cycle of G, we use $Int_G(C)$ to denote the set of vertices located in the interior of C.

Let H be a maximal subgraph of a plane graph, $\{f_1, f_2, \ldots, f_l\}$ the set of faces of H. Suppose that f_1 is the unbounded face of H. We define $link(H) = \sum_{i=2}^{l} b(f_i) - b(f_1)$. Then $link(H)$ is a cycle of $C(G)$.

We use $T(G)$ to denote the set of maximal subgraphs of G which has at least four vertices and contains no separating triangles(a triangle $uvwu$ is a separating triangle if both the interior and the exterior of $uvwu$ contain vertices of G), and let $LT(G) = \{link(H)|H \in T(G)\}$.

One can easily check the following

Lemma 21 *If G is a plane graph, then $H_q(G) = 0$ for $q \geq 3$.*

Lemma 22 *If G is a plane graph, then any three distinct 3-simplexes of $C(G)$ can not share a common 2-simplex.*

Let $x = \sum_{i=1}^{\alpha_q} r_i s_i^q (r_i \in J)$ and $z = \sum_{i=1}^{\alpha_q} s_i s_i^q (s_i \in J)$ be two q-cycles of a complex K. If $|s_i| \leq |r_i| (i = 1, 2, \ldots, \alpha_q)$, $r_i s_i > 0$ while $s_i \neq 0$, and $r_{i_0} \neq s_{i_0}$ for some $i_0 \in \{1, 2, \ldots, \alpha_q\}$, then z is called a *proper subcycle* of x. A *basic cycle* is a cycle containing no proper subcycle.

Lemma 23 *Let G be a plane graph, c a 2-cycle of $C(G)$. Then, c is a basic cycle if and only if $c = link(H)$ for some maximal plane subgraphs $H \ncong K_3$.*

Proof. The sufficiency is clearly. Following we prove the necessity by contradiction. If it is not so, let c be a basic 2-cycle and $c \neq link(H)$ for any maximal plane subgraph $H \neq K_3$. Let H_0 denote the subgraph of G induced by the edges corresponding to the 1-simplexes in c. If H_0 is not a maximal plane graph, then it must has a non-triangular face, say f_0, let u_0v_0 be an edge incident with f_0.

Without loss of generality, assume that f_0 is the unbounded face of H_0. Because c is a cycle, there must be two distinct triangles $x_0u_0v_0x_0$ and $y_0u_0v_0y_0$ in H_0 such that $x_0 \in Int_{H_0}(y_0u_0v_0)$. Let $G_0 = H_0[\{y_0, u_0, v_0\} \cup Int_{H_0}(y_0u_0v_0y_0)]$. Then $|V(G_0)| < |V(G)|$.

If there still exist l-faces in G_0 with $l \geq 4$, then take such a face, say f_1, and an edge u_1v_1 incident with f_1. There must be two distinct triangles $x_1u_1v_1x_1$ and $y_1u_1v_1x_1$ in G_0 such that $x_1 \in Int_{G_0}(y_1u_1v_1)$ because c is a cycle. Let $G_1 = G_0[\{y_1, u_1, v_1\} \cup Int_{G_0}(y_1u_1v_1y_1)]$. Then, $|V(G_1)| < |V(G_0)|$.

Go ahead so on and so forth. Because $V(H_0)$ is finite, there must be a triangle $yuvy$ in H_0 such that $Int_{H_0}(yuvy) \neq \emptyset$ and all faces of H_0 in the interior of $yuvy$ are 3-faces. Let $H' = H_0[\{y, u, v\} \cup Int_{H_0}(yuvy)]$, and let $c' = link(H')$. Clearly, c' is a proper subcycle of c. A contradiction. ∎

Let G_1 and G_2 be two plane graphs, f_1 with boundary $u_1u_2u_3u_1$ and f_2 with boundary $v_1v_2v_3v_1$ be two 3-faces of G_1 and G_2, respectively. We can embed G_2 into the 3-face f_1 by identifying u_i with v_i $(i = 1, 2, 3)$, respectively. This new plane graph is called a *triangle-embedding of* G_2 *in* G_1.

Lemma 24 *Let G be a plane graph, c a basic 2-cycle of $C(G)$. Then,*

$$c \sim \sum_{c \in LT(G)} r(c)c, \text{ where } r(c) = 0, \text{ or } 1. \tag{1}$$

Proof. By Lemma 23, there exists a maximal subgraph $H \neq K_3$ such that $c = link(H)$. Without loss of generality, suppose $H \ncong K_4$ and $H \notin T(G)$, u, w, v are the three vertices incident with the unbounded face of H in clockwise order. The proof is by induction on the order of H. It is obviously that $|V(H)| \geq 5$. While $|V(H)| = 5$, H is isomorphic to the graph obtained by embedded K_4 in K_4. Obviously, (1) holds.

Assume that (1) is valid while $|V(H)| \leq n$ $(n \geq 5)$. Following suppose $|V(H)| = n + 1$.

Case 1. There exists $x \in Int_H(uvwu)$ such that $H[\{u, v, w, x\}] \cong K_4$. Since $H \ncong K_4$, then at least one of the three cycles $xuvx, xuwx$ and $xvwx$ is nonempty. Without loss of generality, suppose $xuvx, xuwx, xvwx$ are all nonempty.

Let $c_{uv} = link(H[\{x, u, v\} \bigcup Int_H(xuvx)]$, $c_{vw} = link(H[\{x, v, w\} \bigcup Int_H(xvwx)]$ and $c_{uw} = link(H[\{x, u, w\} \bigcup Int_H(xuwx)]$. Suppose abc is a simplex of c, $q, q_{uv}, q_{vw}, q_{uw}$ denote the coefficients of abc in $c, c_{uv}, c_{vw}, c_{uw}$, respectively. Then, by the definition of $link(H)$,

$$q = q_{uv} = 1, q_{vw} = q_{uw} = 0, \text{ while } \{a, b, c\} \subset Int_H(xuvx), \tag{2}$$

$$q = q_{vw} = 1, q_{uv} = q_{uw} = 0, \text{ while } \{a, b, c\} \subset Int_H(xvwx), \tag{3}$$

$$q = q_{uw} = 1, q_{vw} = q_{uv} = 0, \text{ while } \{a, b, c\} \subset Int_H(xuwx), \tag{4}$$

Since 2-simplex uwv appears only in c, xvu appears only in c_{uv}, xwv only in c_{vw}, and xuw appears only in c_{uw}. Combining (2), (3) and (4), we have

$$c - c_{uv} - c_{vw} - c_{uw} = -uwv + xwv + xuw + xvu = \partial xuvw,$$

i.e., $c \sim c_{uv} + c_{vw} + c_{uw}$. (1) holds by the induction hypothesis.

Case 2. There exists separating triangle $abca \in H$. Let $H_0 = H[\{a, b, c\} \bigcup Int_H(abca)]$, $H_1 = H[V(H) - Int_H(abca)]$. Then, $c = link(H_0) + link(H_1)$, hence (1) holds by the induction hypothesis. ∎

Lemma 25 *Let G be a plane graph. Then chains in $LT(G)$ are not boundary chains and any two chains in $LT(G)$ are not homologous.*

Proof: By contradiction. Let $H \in T(G)$, $c = link(H) = \partial z$ where z is a 3-chain of $C(G)$. Without loss of generality, suppose $x_0u_0v_0w_0$ is a simplex of z such that $x_0 \in Int_G(u_0v_0w_0u_0)$ and

$$|Int_G(u_0v_0w_0u_0)| = \min\{|Int(uvwu)||\exists x \in Int_G(uvwu), xuvw \text{ is a simplex of } z\}.$$

By the definition of $T(G)$, H contains no separating triangle, so $x_0u_0v_0$ is not a simplex of c, and hence there exists $y_0 \in Int_G(x_0u_0v_0x_0) - V(H)$ such that $x_0u_0v_0y_0$ is a simplex of z. It is clearly that $|Int_G(x_0u_0v_0x_0)| < |Int_G(u_0v_0w_0u_0)|$, contradicts to the choice of $u_0v_0w_0u_0$. ∎

Lemma 26 *Let G be a connected plane graph. Then, $H_0(G) = J$, $H_1(G) = [(m+1) - \chi(C(G))]J$, $H_2(G) = mJ$, where $m = |T(G)|$.*

Proof: Because G is connected, then $C(G)$ is connected, by Lemma 24, Lemma 25, $H_0(G) = J$, and $H_2(G) = mJ$. Because each basic 1-cycle of $C(G)$ corresponds to a unique cycle of G, then $C(G)$ has no 1 dimensional *torsion coefficient*, and so, $H_1(G) = [(m+1) - \chi(C(G))]J$. ∎

A *pseudo-maximal plane graph* is a 2-connected plane graph which contains at most one non-triangular face.

Lemma 27 *Let G be a 2-connected plane graph such that $H_1(G) = 0$, and f a non-triangular face of G. Then, there exists a pseudo-maximal subgraph H of G such that f is the unique non-triangular face of H.*

Proof: Let $C = v_1v_2 \dots v_nv_1$ $(n \geq 4)$ be the boundary of f, $c = \sum_{i=1}^{n} v_iv_{i+1} \pmod n$.

Let $\{s_i^2 | i = 1, 2, \dots, \alpha_2\}$ be the set of 2-simplexes of $C(G)$. Since $H_1(G) = 0$, then there exists a 2-chain of $C(G)$, say $x = \sum_{i=1}^{\alpha_2} a_i s_i^2$, such that $c = \partial x$. We chose such a 2-chain with the property that

$$\sum_{i=1}^{\alpha_2} |a_i| = \min\{\sum_{i=1}^{\alpha_2} |b_i| | c = \partial \sum_{i=1}^{\alpha_2} b_i s_i^2\}. \tag{5}$$

Let H be the subgraph induced by those edges corresponding to the 1-simplexes contained in x. Let $H^* = H + \{uv_i\}_{i=1}^n$, $G^* = G + \{uv_i\}_{i=1}^n$, where $u \notin V(G)$.

Clearly, G^* is a plane graph, H^* is a subgraph of G^*, and $y = x + \sum_{i=1}^{n} uv_{i+1}v_i \pmod n$ is a 2-cycle of $C(G^*)$. According to (5), y has no proper subcycles, i.e., y is a basic cycle. By lemma 23, H^* is a maximal plane graph, therefore f is the unique non-triangular face of H. ∎

Proof of Theorem 12: We only need to prove the necessity. By definition 11, we can assume that G is 2-connected. By Lemma 26 and Lemma 27, $|T(G)| = 0$

and G can be obtained by a serial of triangle-embeddings of its pseudo-maximal plane subgraphs. Since contractible transformations are inversable, it suffices to show that each pseudo-maximal plane subgraph of G with acyclic clique complex is contractible.

Let B be a pseudo-maximal plane subgraph of G with acyclic clique complex. Without loss of generality, suppose that $B \ncong K_3$. It is easy to verify that B is contractible while $|E(B)| \leq 8$. Now we proceed to the induction process.

Suppose each pseudo-maximal plane graph with less than n edges and with acyclic clique complex is contractible, where $n \geq 9$. Let B be a pseudo-maximal plane subgraph of G with acyclic clique complex, $|E(B)| = n$, C the boundary of the non-triangular face of B, and uv an edge of C. Then we have

$$|N_B(u) \cap N_B(v)| = 1. \tag{6}$$

If it is not so, let $x_1, x_2 \in N_B(u) \cap N_B(v)$ and $Int_B(x_1uvx_1) \cap V(B) = \emptyset$.

Case 1. $x_1x_2 \in E(B)$. Clearly, $G[\{x_1, x_2, u, v\}] \cong K_4$. If $Int_G(x_1x_2vx_1) \cup Int_H(x_1x_2$
$ux_1) = \emptyset$, then $d_B(x_1) = d_G(x_1) = 3$. By the definition of contractible transformations, x_1 can be deleted from B, and by the induction hypothesis, B is contractible. So we assume $Int_G(x_1x_2vx_1) \cup Int_G(x_1x_2ux_1) \neq \emptyset$. Go on and go forth, one can always find a vertex $x_0 \in Int_G(uvx_2u)$ such $d_G(x) = 3$ and $\{x\} \cup N_B(x)$ induced a K_4, then x_0 can be deleted from B, and so B is contractible by the induction hypothesis.

Case 2. $x_1x_2 \notin E(B)$. There must be a vertex in $Int_B(ux_1vx_2u)$ which is adjacent to x_1 because B is a pseudo-maximal plane subgraph. Therefore, either there exists a 3-vertex a in B such that $B[\{a\} \cup N_B(a)] \cong K_4$ which indicates that deletion of a from B is a contractible transformation, or B contains a subgraph which is in $T(G)$ which contradicts to that $C(G)$ is acyclic.

Now, (6) is proved. Deletion of uv from B is a contractible transformation. Let $B_1 = B - uv$. If $\delta(B_1) = 1$, then one of u or v, say u, is a 2-vertex in B, so deletion of u from B is a contractible transformation, and hence by the induction hypothesis, B is contractible. If $\delta(B_1) \geq 2$, then B_1 is a pseudo-maximal plane subgraph of B and $|E(B_1)| = n - 1$, again by the induction hypothesis, B_1 is contractible. ∎

References

1. Bondy J A, Murty U S R. Graph theory with applications. New York: Macmillan Ltd. Press, 1976.
2. Jiang C.H.. Introduction to Topology. Shanghai:Shanghai Publishing Company of Science and Technology, 1978(In Chinese).
3. V.A.Kovalevsky. Finite topology as applied to image analyses. Comput. Vision Graphics Image Process 46(1989)141-161.
4. A. V. Ivashchenko. Contractible transformations do not change the homology groups of graphs. Discrete Math., Vol.126(1994):159-170.

On the Domination Numbers of Generalized de Bruijn Digraphs and Generalized Kautz Digraphs (Extended Abstract)

Yosuke Kikuchi and Yukio Shibata

Department of Computer Science, Gunma University,
Kiryu, Gunma 376-8515, Japan
{kikuchi, shibata}@msc.cs.gunma-u.ac.jp

Abstract. This work deals with the domination numbers of generalized de Bruijn digraphs and generalized Kautz digraphs. Dominating sets for digraphs are not familiar compared with dominating set for undirected graphs. Whereas dominating sets for digraphs have more applications than undirected graphs. We construct dominating sets of generalized de Bruijn digraphs under some conditions. We investigate consecutive minimum dominating set of the generalized de Bruijn digraphs. For generalized Kautz digraphs, there is a consecutive minimum dominating set that is a minimum dominating set.

1 Introduction

It is one of major areas in theoretical and algorithmic observation to study domination and its related topics for undirected graphs [1,11]. Besides domination has been defined in digraphs, concepts of solution and kernel which are peculiar to digraphs have been defined. The origin of solution and kernel is in game theory [10]. Solution and kernel have applications for logic and facility location [4,7,9,16]. Considering applications of domination and its related topics for game theory and logic, digraphs are more natural and have more applications than undirected graphs. Although digraphs have many applications, there are few studies for domination in digraphs. Barkauskas and Host, and Bar-Yehuda and Vishkin [2,3] have studied algorithms for domination of digraphs. Barkauskas and Host showed that the problem of determining whether or not an arbitrary digraph has an efficient dominating set is NP-complete and gave a linear time algorithm for finding efficient dominating sets in directed trees [2]. There are some researches for kernel in the digraph with no odd circuits [6] and for domination in tournament that is a class of digraphs [8,14].

Each edge has orientation in digraphs, these edges are called arcs. This situation cause difficulty of research for domination in digraphs.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 400–408, 2001.
© Springer-Verlag Berlin Heidelberg 2001

2 Definitions and Terminologies

$V(G)$ and $A(G)$ are the vertex set and the arc set of a digraph $G(V, A)$, respectively. There is an arc from x to y if $(x, y) \in A(G)$. The vertex x is called a *predecessor* of y and y is called a *successor* of x. The sets $O(u) = \{v|(u, v) \in A(G)\}$ and $I(u) = \{v|(v, u) \in A(G)\}$ are called the *outset* and the *inset* of the vertex u, respectively. Similarly, we define $O[u] = O(u) \cup \{u\}$, $I[u] = I(u) \cup \{u\}$. If $S \subset V(G)$ then $O(S) = \cup_{s\in S} O(s)$, $I(S) = \cup_{s\in S} I(s)$, $O[S] = \cup_{s\in S} O[s]$, and $I[S] = \cup_{s\in S} I[s]$. A set $S \subset V(G)$ is a *dominating set* of a digraph G if v is a successor of some vertex $u \in S$ for all $v \notin S$. A set $S \subset V(G)$ is an *absorbant* of a digraph G if there is a vertex $u \in S$ such that u is a successor of v for all $v \notin S$. A set $S \subset V(G)$ is *independent* if for any $x, y \in S$,$(x, y) \notin A(G)$. If $S \subset V(G)$ is independent and a dominating set of a digraph G, S is called a *solution* of G. If $S \subset V(G)$ is independent and an absorbant of a digraph G, S is called a *kernel* of G [10]. $|X|$ denotes the cardinality of a set X. For a vertex $u \in V(G)$, $|O(u)|$ is called the *out degree* of u. We use the notation $\gamma(G)$ for the minimum cardinality of a set $S \subset V(G)$ which is a dominating set. If S is a dominating set of G whose cardinality is equal to $\gamma(G)$, S is called the *minimum dominating set* of G. Let m, n be positive integers. (m, n) denotes the greatest common divisor of m and n. $m|n$ and $m \not| n$ mean that m divides n and m does not divide n, respectively.

In this paper, we will describe domination numbers of generalized de Bruijn digraphs and generalized Kautz digraphs. Furthermore we construct minimum dominating sets of generalized de Bruijn digraphs and generalized Kautz digraphs. Definitions of generalized de Bruijn digraphs and generalized Kautz digraphs are as follows. Generalized de Bruijn digraph $G_B(n, d)$ is defined by congruence equations.

$$\begin{cases} V(G_B(n,d)) = \{0, 1, 2, \ldots, n-1\}, \\ A(G_B(n,d)) = \{(x,y)|y \equiv dx + i \pmod n, 0 \le i < d\}. \end{cases}$$

If $n = d^D$, $G_B(n, d)$ is the de Bruijn digraph $B(d, D)$.

Whereas definition of generalized Kautz digraphs $G_K(n, d)$ is

$$\begin{cases} V(G_K(n,d)) = \{0, 1, 2, \ldots, n-1\}, \\ A(G_K(n,d)) = \{(x,y)|y \equiv -dx - i \pmod n, 0 < i \le d\}. \end{cases}$$

If $n = d(d-1)^{D-1}$, $G_K(n, d)$ is the Kautz digraph $K(d-1, D)$. Generalized de Bruijn digraphs and generalized Kautz digraphs are introduced by Imase and Itoh [12,13], and Reddy, Pradhan and J. Kuhl [15], so called Imase Itoh digraphs. It is well known that de Bruijn digraphs and Kautz digraphs have good property for network topology compared with hypercubes [5]. These generalizations remove the restriction for the cardinality of vertex sets of these digraphs and make these digraphs more valuable for network models.

3 Domination Numbers of Generalized de Bruijn Digraphs

We prepare a simple lemma for determining domination number of generalized de Bruijn digraphs.

Lemma 1. $\gamma(G_B(n,d)) \geq \left\lceil \dfrac{n}{d+1} \right\rceil$.

Proof. Let S be a minimum dominating set of $G_B(n,d)$.We obtain $|S|+d|S| \geq n$ from the definition of $G_B(n,d)$.

Above lemma gives the lower bound for the domination number of $G_B(n,d)$. We also obtain the following lemma concerning the upper bound of the domination number of $G_B(n,d)$.

Lemma 2. $\gamma(G_B(n,d)) \leq \left\lfloor \dfrac{n+1}{d} \right\rfloor$.

Proof. For $G_B(n,d)$, let $S = \{1,2,\ldots,\lfloor (n+1)/d \rfloor\}$. Then $O(S) = \{d, d+1, d+2, \ldots, \lfloor (n+1)/d \rfloor\, d + d - 1\}$ with duplications. $|O(S)| = \lfloor (n+1)/d \rfloor\, d + d - 1 = (\lceil (n+1)/d \rceil - 1)\, d + d - 1 \geq n$. Thus S is a dominating set of $G_B(n,d)$. □

From the proof of Lemma 2, $S = \{1,2,\ldots,\lfloor (n+1)/d \rfloor\}$ is a dominating set, but not minimum. For example, the out set of a subset $\{1,2,3,4\}$ of $G_B(12,2)$ is $\{2,3,4,5,6,7,8,9\}$. Then $\{1,2,3,4\}$ is not a dominating set of $G_B(10,4)$. Whereas $\{1,2,3,4,5,6\}$ is a dominating set of $G_B(10,4)$, still $\gamma(G_B(12,2)) = 4$ because the vertex subset $\{4,5,6,7\}$ is a minimum dominating set of $G_B(12,2)$. Thus $\{1,2,3,4,5,6\}$ is not a minimum dominating set. Next theorem shows a sufficient condition for the domination number of $G_B(n,d)$ to be $\lceil n/(d+1) \rceil$ and a method of determining the dominating set of $G_B(n,d)$.

Theorem 1. *If there is a vertex $x \in V(G_B(n,d))$ satisfying,*

$$(d-1)x - \left(\left\lceil \frac{n}{d+1} \right\rceil - l \right) \equiv 0 \pmod{n} \tag{1}$$

for some l $(0 \leq l \leq (d+1)\lceil n/(d+1) \rceil - n)$, then

$$\gamma(G_B(n,d)) = \left\lceil \frac{n}{d+1} \right\rceil.$$

Proof. From Lemma 1, we show the correctness of this statement by constructing a dominating set of $G_B(n,d)$ that contains x satisfying equation (1). Let $D = \{x, x+1, \ldots, x + \lceil n/(d+1) \rceil - 1\} \subseteq V(G_B(n,d))$. Then out sets of vertices in D are as follows.

$$\begin{aligned}
&O(x) = \{dx, dx+1, \ldots, dx+d-1\},\\
&O(x+1) = \{d(x+1), d(x+1)+1, \ldots, d(x+1)+d-1\},\\
&\quad\vdots\\
&O(x + \lceil n/(d+1) \rceil - 1) = \{d(x + \lceil n/(d+1) \rceil - 1), d(x + \lceil n/(d+1) \rceil - 1) + 1,\\
&\qquad\qquad \ldots, d(x + \lceil n/(d+1) \rceil - 1) + d - 1\}.
\end{aligned}$$

Thus $|O(D)| = d\left\lceil \frac{n}{d+1} \right\rceil$, and $|D| = \left\lceil \frac{n}{d+1} \right\rceil$. Since x satisfies equation (1),

$$dx \equiv x + \lceil n/(d+1) \rceil - l \pmod{n}.$$

Therefore $|D \cap O(x)| = l$, and by the condition for l, we obtain $\lceil n/(d+1) \rceil - l \geq 0$.

$$\begin{aligned} &|D| + |O(D)| - |D \cap O(x)| \\ &\quad = \lceil n/(d+1) \rceil + d\lceil n/(d+1) \rceil - l \\ &\quad \geq \lceil n/(d+1) \rceil + d\lceil n/(d+1) \rceil - \{(d+1)\lceil n/(d+1) \rceil - n\} \\ &\quad = n. \end{aligned}$$

Hence D is a minimum dominating set of $G(n,d)$. □

We obtain the following statement for a special case Theorem 1.

Corollary 1. *For $G_B(n,d)$, $\gamma(G_B(n,d)) = 1$ if and only if $d \geq n$ or n is odd and $d = n-1$.*

Proof. We only show the necessary condition because sufficiency is obvious. Since $\gamma(G_B(n,d)) = 1$, there is a vertex $x \in V(G_B(n,d))$ such that $|O[x]| \geq n$. If $|O(x)| \geq n$ then $d \geq n$ because of the definition of $G_B(n,d)$. If $|O(x)| = n-1$ then $d = n-1$ and $x \in V(G_B(n,d))$ has no loop. Hence, we obtain $x+1 \equiv (n-1)x \pmod{n}$. Therefore $2x + 1 \equiv 0 \pmod{n}$, then n is odd. □

From Theorem 1, we see the following corollary for the solution of $G_B(n,d)$.

Corollary 2. *Let $d+1|n$. Then if there is a vertex $x \in V(G_B(n,d))$ such that*

$$(d-1)x - \frac{n}{d+1} \equiv 0 \pmod{n},$$

then there is a solution S of $G_B(n,d)$ such that

$$|S| = \frac{n}{d+1}.$$

We prepare the *reversal* of a digraph G, denoted by G^{-1}. The reversal of a digraph G is defined by $V(G^{-1}) = V(G)$, $(x,y) \in A(G^{-1})$ if and only if $(y,x) \in A(G)$. For generalized de Bruijn digraphs, $G_B(n,d)$ is not necessarily isomorphic to $G_B(n,d)^{-1}$. For de Bruijn digraph $B(d,D)$, we can see that $B(d,D) \cong B(d,D)^{-1}$. Then there is one to one correspondence between the dominating set and the absorbant in $B(d,D)$. For $B(d,D)$, the minimum cardinality of an absorbant is equal to the domination number. Therefore, we obtain the following statement.

Corollary 3. *For $B(d,D)$, both the minimum cardinalities of the absorbant and the dominating set are $\left\lceil \frac{d^D}{d+1} \right\rceil$.*

Proof. We first assume that D is even. Then the value l in Theorem 1 takes an integer between 0 to d and from the definition of $B(d, D)$, $d \geq 2$. We put

$$x \equiv d^{D-2} + d^{D-4} + \cdots + d^{D-2k} + \cdots + d^2 + 1 \pmod{d^D},$$
$$l = 1,$$

then x satisfies equation (1) in Theorem 1. Next let us assume that D is odd, then the value l in Theorem 1 takes an integer between 0 to l. We put

$$x \equiv d^{D-2} + d^{D-4} + \cdots + d^{D-2k} + \cdots + d^3 + d \pmod{d^D},$$
$$l = 1,$$

then x satisfies equation (1) in Theorem 1. □

Minimum dominating set is not necessarily a set of vertices whose labels are consecutive. For $G_B(10, 5)$, both $\{1, 2\}$ and $\{3, 6\}$ are minimum dominating sets of $G_B(10, 5)$. The set $\{1, 2\}$ is constituted of consecutive vertices and the set $\{3, 6\}$ is not constituted of consecutive vertices. Thus Theorem 1 gives a sufficient condition but not a necessary condition. It is not easy to find all minimum dominating sets of $G_B(n, d)$. We can not determine that domination number of a given $G_B(n, d)$ is equal to $\lceil n/(d+1) \rceil$ only using Theorem 1 because $G_B(n, d)$ may have a minimum dominating set which is constituted of non-consecutive vertices and whose cardinality is equal to $\lceil n/(d+1) \rceil$. Then we give a necessary and sufficient condition for $G_B(n, d)$ to have a minimum dominating set S with $|S| = \lceil n/(d+1) \rceil$ and constituted of consecutive vertices. We call such S a *consecutive minimum dominating set.*

Theorem 2. *$G_B(n, d)$ satisfies one of the following four conditions if and only if there is a vertex $x \in V(G_B(n, d))$ such that $\{x, x+1, \ldots, x+\lceil n/(d+1) \rceil - 1\}$ is a consecutive minimum dominating set.*
Conditions
(*i*) $d+1 \not| n$,
(*ii*) $d+1 | n$ *and* d *is even*,
(*iii*) $2(d+1) | n$ *and* d *is odd and* $4 | d+1$,
(*iv*) $2^k(d+1) | n$ *and* d *is odd and* $d+1$ *contains only one 2 as a factor.*
($d - 1 = 2^k s$, s *is odd*)

Proof. It is obvious that $\gamma(G_B(n, d)) = 1$, when $d \geq n$. Suppose that $d < n$. We show that there is a vertex x in $V(G_B(n, d))$ such that

$$x + \lceil n/(d+1) \rceil \equiv dx + i \pmod{n}, \tag{2}$$
$$0 \leq i \leq \min\{d - 1, \lceil n/(d+1) \rceil (d+1) - n\}. \tag{3}$$

From equation (2),we obtain

$$(d-1)x \equiv \lceil n/(d+1) \rceil - i \pmod{n}. \tag{4}$$

For equation (3), the value i takes an integer between 0 and $d-1$ if $d+1$ is not a divisor of n. If $d+1 | n$, then $i = 0$. $G_B(n, d)$ has a consecutive dominating set S

with $|S| = \lceil n/(d+1) \rceil$ if and only if there is a vertex $x \in V(G_B(n,d))$ satisfying equation (4). If $d+1$ is not a divisor of n, let $(d-1,n) = s$. Then there are q and r, $(0 \leq r < s)$ such that $\lceil n/(d+1) \rceil = qs + r$. We can put $r = i$, from the ranges of r and i. Thus there are x and i satisfying equations (2) and (3) in Z_n. Next, if $d+1$ is a divisor of n,

$$(d-1)x \equiv n/(d+1) \pmod{n} \tag{5}$$

by equation (4) . The equation (5) has a solution if and only if

$$((d-1),n) | n/(d+1). \tag{6}$$

Hence if d is even, there is an integer x satisfying equation (5) in Z_n.
Whereas if d is odd, $d-1$,$d+1$ and n are even integers. We put $d+1 = 2^h\alpha$ ($h(\leq 2)$ is an integer,α is an odd integer),then $d-1 = 2(2^{h-1}\alpha - 1)$. We obtain $2|d-1$ and $4 \not| d-1$, since $2^{h-1}\alpha - 1$ is odd. Therefore n is divided by 2^{h+1}. Then n is divided by $2(d+1)$.
For $d+1 = 2^h\alpha$, we consider $h = 1$, then put $d-1 = 2(\alpha-1) = 2^k\beta$, ($\beta$ is odd). 2^{h+1} divides n by (6). Therefore n is divided by $2^h(d+1)$. Conversely if $G_B(n,d)$ has a consecutive minimum dominating set $\{x, x+1, \ldots, x+\lceil n/(d+1) \rceil - 1\}$, then x satisfies equation (4). Therefore n and d satisfy one of above four conditions. □

We consider whether there exists $G_B(n,d)$ that does not have a minimum consecutive dominating set and $\gamma(G_B(n,d)) = \lceil n/(d+1) \rceil$. We show that if $\lceil n/(d+1) \rceil = 1, 2$, or 3, there does not exist $G_B(n,d)$ that does not have a minimum dominating set and $\gamma(G_B(n,d)) = \lceil n/(d+1) \rceil$.

Theorem 3. *For domination number of generalized de Bruijn digraphs, we obtain*

$$\gamma(G_B(2s, 2s-1)) = 2, \tag{7}$$
$$\gamma(G_B(8s-4, 4s-3)) = 3, \tag{8}$$

and

$$\gamma(G_B(6s, 2s-1)) = 4, \tag{9}$$

where s is a natural number.

Proof. For equation(7), each vertex of $G_B(2s, 2s-1)$ has a loop. Then $\gamma(G_B(2s, 2s-1)) \neq 1$
For equation(8), let $\gamma(G_B(8s-4, 4s-3)) = 2$ and a vertex x is contained in a dominating set of $G_B(8s-4, 4s-3)$. Then $x+1$ or $x+4s-2$ is contained in the dominating set of $G_B(8s-4, 4s-3)$. If the dominating set is $\{x, x+1\}$, then

$$(4s-3)x \equiv x+2 \pmod{8s-4},$$
$$4(s-1)x \equiv 2 \pmod{8s-4}.$$

Therefore we have a contradiction.
If the dominating set is $\{x, x+4s-2\}$, then we obtain

$$(4s-3)x \equiv x+1 \pmod{8s-4} \tag{10}$$

or

$$(4s-3)(x+4s-2) \equiv x+1 \pmod{8s-4}. \tag{11}$$

If equation (10) holds, then $(4s-4)x \equiv 1 \pmod{8s-4}$. This is a contradiction. If equation (11) holds, then $(4s-4)x \equiv 4(5s-4s^2)-5 \pmod{8s-4}$. This is also a contradiction.
For equation(9), let us assume that $\gamma(G_b(6s, 2s-1)) = 3$. We should consider three cases ; (1) any two vertices are not consecutive in the dominating set, (2) exactly two vertices are consecutive in the dominating set, and (3) vertices in the dominating set are consecutive. Case (1); let a vertex x be in the dominating set, then other vertices in the dominating set are $x+2s$ and $x+4s$. Thus x satisfies

$$(2s-1)x \equiv x+2ts+1 \pmod{6s}$$

for some t $(t = 0, 1, 2)$. From this equation, we obtain

$$2(s-1)x \equiv 2ts+1 \pmod{6s},$$

and this is a contradiction. Case (2); let x and $x+1$ be consecutive vertices in the dominating set, then another vertex in the dominating set is $x+2s+1$ or $x+4s$. If the dominating set is $\{x, x+1, x+2s+1\}$, then

$$\begin{cases} (2s-1)(x+2s+1) \equiv x+2 \pmod{6s}, \\ (2s-1)(x+1)+2s-2 \equiv x-1 \pmod{6s}. \end{cases}$$

By these simultaneous equations, we obtain

$$4s(s-1) \equiv 1 \pmod{6s}.$$

Then this is a contradiction.
If the dominating set is $\{x, x+1, x+4s\}$ then

$$(2s-1)(x+4s)+2s-2 \equiv x-1 \pmod{6s}.$$

By this equation, we obtain

$$2(s-1)x \equiv 8s-8s^2+1 \pmod{6s}$$

and this is a contradiction.
Case (3); let x, $x+1$ and $x+2$ constitute a dominating set. Then

$$(2s-1)x \equiv x+3 \pmod{6s}.$$

Hence we obtain

$$2(s-1)x \equiv 3 \pmod{6s},$$

and this is also a contradiction. □

4 Domination Numbers of Generalized Kautz Digraphs

We can see the following lemma similar to Lemma 1.

Lemma 3. $\gamma(G_K(n,d)) \geq \left\lceil \frac{n}{d+1} \right\rceil$. □

Besides the difficulty of determining domination numbers of generalized de Bruijn digraphs, it is easy to determine the domination number of a generalized Kautz digraph $G_K(n,d)$.

Theorem 4.

$$\gamma(G_K(n,d)) = \left\lceil \frac{n}{d+1} \right\rceil$$

Proof. $D = \{0,1,2,\ldots,\left\lceil \frac{n}{d+1} \right\rceil - 1\} \subseteq V(G_K(n,d))$ is a dominating set of $G_K(n,d)$ □

5 Conclusion

We investigated domination numbers of generalized de Bruijn digraphs and generalized Kautz digraphs. Consecutive dominating set is important in our investigations. Finding a dominating set of a generalized de Bruijn digraph is more difficult than that of generalized Kautz digraph. We conjecture that there is a consecutive minimum dominating set for $G_B(n,d)$.

Acknowledgment

The authors would like to thank anonymous referees for their helpful suggestions.

References

1. Allan, R. B., Laskar, R.: On domination and independent domination number of a graph. Discrete. Math. **23** (1978) 73–76
2. Barkauskas, A. E., Host, L. H.: Finding efficient dominating sets in oriented graphs. Congr. Numer. **98** (1993) 27–32
3. Bar-Yehuda, R., Vishkin, U.: Complexity of finding k-path-free dominating sets in graphs. Info. Process. Letter **14** (1982) 228–232
4. Berge, C., Rao, A. R.: A combinatorial problem in logic. Discrete. Math. **17** (1977) 23–26
5. Bermond, J.-C. Peyrat, C.: De Bruijn and Kautz networks : a competitor for the hypercube ?. Hypercube and Distributed Computers (F. Andrè and J. P. Verjus, Eds.). Elsevier North-Holland, Amsterdam, 1989
6. Chaty, G., Szwarcfiter, J. L.: Enumerating the kernels of a directed graph with no odd circuits. Info. Proc. Letter **51** (1994) 149–153
7. Duchet, P., Meyniel, H.: Kernels in directed graphs: a poison game. Discrete. Math. **115** (1993) 273–276

8. Fisher, D., Lundgren, J. R., Merz, S. K., Reid, K. B.: Domination graphs of tournaments and digraphs. Congr. Numer. **108** (1995) 97–107
9. Fraenkel, A. S., Yesha, Y.: Complexity of problems in games,graphs, and algebraic equations. Discrete. Appl. Math. **1** (1979) 15–30
10. Ghoshal, J., Laskar, R., Pillone, D.: Topics on domination in directed graphs. Domination in graphs (T. W. Haynes, S. T. Hedetniemi and P. J. Slater, Eds.). Marcel Dekker, New York, 1998, 401–437
11. Haynes, T. W. Hedetniemi, S. T. Slater, P. J.: Fundamentals of Domination in Graphs. Marcel Dekker, New York, 1998
12. Imase, M., Itoh, M.: Design to minimize diameter on building-block network. IEEE Trans. Computer **C-30** (1981) 439–442
13. Imase, M., Itoh, M.: A design for directed graphs with minimum diameter. IEEE Trans. Computer **C-32** (1983) 782–784
14. Megiddo, N., Vishkin, U.: On finding a minimum dominating set in a tournament. Theor. Comp. Sci. **61** (1988) 307–316
15. Reddy, S. M., Pradhan, D. K., Kuhl, J.: Directed graphs with minimal diameter and maximum node connectivity. School of Engineering Oakland Univ. Tech. Report , 1980
16. von Neumann J., Morgenstern, O.: Theory of Games and Economic Behaviour. Princeton University Press, Princeton, 1944.

A Notion of Cross-Perfect Bipartite Graphs

Milind Dawande

The University of Texas at Dallas and IBM T. J. Watson Research Center, NY,
milind@utdallas.edu

Abstract. In this note, we consider four quantities defined on a bipartite graph B: the cross-chromatic index $\chi^*(B)$, the biclique number $w^*(B)$, the cross-free matching number $m^*(B)$ and the biclique edge covering number $\beta^*(B)$. We mention several applications of these numbers and define a notion of cross-perfect bipartite graphs. A duality between these numbers for the class of cross-perfect graphs is examined.

Keywords: bipartite graph; perfect graph; integral polytope.

1 Definitions

Given a bipartite graph $B = (U \cup V, E)$, two non-adjacent edges $e, e^{'} \in E$ with $e = (u_1, v_1)$ and $e^{'} = (u_2, v_2)$ are said to form a *cross* if $(u_1, v_2) \in E$ and $(u_2, v_1) \in E$. Two edges are said to be *cross-adjacent* if either they are adjacent (i.e. share a common node) or they form a cross. A *cross-free matching* in B is a set of edges $E^{'} \subseteq E$ with the property that no two edges $e, e^{'} \in E^{'}$ are cross-adjacent. A *cross-free coloring* of B is a coloring of the edge set E subject to the restriction that no pair of cross-adjacent edges has the same color. A set of edges $E^{'} \subseteq E$ is said to form a *biclique* if the induced subgraph corresponding to $E^{'}$ is a complete bipartite subgraph of B. A *biclique edge cover* for B is a covering of the edge set E by bicliques.

We now consider the following four quantities on B: (i) The *cross-chromatic index*, $\chi^*(B)$, of B which is the minimum number of colors required to get a cross-free coloring of B (ii) The *biclique number*, $w^*(B)$, of B defined as the cardinality of the maximum edge biclique in B (iii) The *cross-free matching number* of B, $m^*(B)$, defined as the edge cardinality of the maximum cross-free matching in B and (iv) The *biclique edge covering number* of B, $\beta^*(B)$, defined as the minimum number of bicliques required to cover the edges of B.

This note is concerned with a duality between the four numbers defined above. The duality, as stated here, resembles the classical duality between the four quantities, namely, stability number $\alpha(G)$, clique partition number $\theta(G)$, chromatic number $\gamma(G)$ and the maximum clique number $\omega(G)$ for a perfect graph G [1][2][9]. In fact, we show that the duality between $w^*(B)$, $\chi^*(B)$, $m^*(B)$ and $\beta^*(B)$ can be studied by investigating when a certain class of graphs is perfect.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 409–413, 2001.
© Springer-Verlag Berlin Heidelberg 2001

2 Applications and Complexity Results

The problem of finding a maximum cross-free matching in a bipartite graph is discussed in [16]. This problem was studied as a special case of a more general problem of finding *alternate cycle free matchings* in bipartite graphs. Alternate cycle free matchings were first investigated in [10] in the context of dependence graphs of matroids: If $\mathcal{M}$ is a matroid with basis B, finding a maximum cardinality alternating cycle free matching is related to finding a basis of $\mathcal{M}$ for which the intersection with B is of minimum cardinality. Another application studied in [3] and mentioned in [16] concerns the computing of the *setup number* or *step number* $\nu(G)$ of an acyclic directed graph G.

The problem of finding a maximum edge cardinality biclique has applications in manufacturing in the computer industry. This application is discussed in detail in [5]. In short, the relationship between a set of products and a set of components which constitute the products can be represented by a bipartite graph. One way to reduce manufacturing lead times is to reduce the final assembly times for the products by creating sub-assemblies. Finding good sub-assemblies in this context can be done by finding large edge-cardinality bicliques in the bipartite graph. Another interest ing application, also described in [5], occurs in the area of formal concept analysis [7].

Covering the edges of a bipartite graph by bicliques has an interesting application [4] is combinatorial chemistry: *Split Synthesis* [6][11], is a method of choice to build libraries for combinatorial chemistry. The basis for this technology is that molecules can be grown with one end tethered to a resin bead, while the other end is extended one residue at a time in parallel across a set of beads. Split synthesis can be modeled by a directed acyclic graph in which each node represents a grow step on a particular subset of beads. In [4], it is shown that optimizing split synthesis construction is intimately related to covering the edges of a bipartite graph by bicliques.

Finding $\beta^*(B)$ is NP-hard [8]. In [16], it is shown that finding $m^*(B)$ is NP-hard. Recently, R. Peeters [15] showed that finding $w^*(B)$ is also NP-hard. To the best of our knowledge, the complexity of finding $\chi^*(B)$ is open.

3 Formulations and Results

Clearly, a biclique cannot have more than one edge from a cross-free matching. This immediately gives the following integer programming formulation for $w^*(B)$

$$w^*(B) = \max\{1x : K^*x \leq 1, x \in \{0,1\}^n\}$$

where $n = |E|$ and K^* is a (0, 1) incidence matrix whose rows correspond to maximal cross-free matchings of B and whose columns correspond to the edges of B. It is also easy to see that

$$\chi^*(B) = \min\{1y : yK^* \geq 1, y \in \{0,1\}^m\}$$

The *fractional biclique polytope* of B is $P = \{x \in R_+^n : K^*x \leq 1\}$. We denote the convex hull of feasible integer solutions of P by P_I. We generalize the definitions of w^* and χ^* to consider binary objective functions. For $c \in \{0,1\}^n$

$$w^*(B_c) = \max\{cx : K^*x \leq 1, x \in \{0,1\}^n\}$$

$$\chi^*(B_c) = \min\{1y : yK^* \geq c, y \in \{0,1\}^m\}$$

Note that the definitions of $w^*(B)$ and $w^*(B_c)$ differ only in the objective function. The constraint matrix K^* is the same for both. Also, $w^*(B_c)$ does *not* correspond to the optimal edge cardinality biclique in the subgraph (either edge-induced or node-induced) corresponding to the vector c. This is an important distinction between the definition of $w^*(B_c)$ and the corresponding definition for the maximum clique in a graph: For a graph G, the formulation for the maximum clique, $\alpha(G)$, generalized to binary objective functions, can be written as

$$\alpha(G_c) = \max\{cx : Kx \leq 1, x \in \{0,1\}^p\}$$

where $p = |V(G)|$ and K is the clique matrix of G [14]. Here, $\alpha(G_c)$ corresponds to the maximum clique on the node-induced subgraph of G corresponding to c. A similar observation can be made about $\chi^*(B_c)$.
Definition: B is cross-perfect if $w^*(B_c) = \chi^*(B_c)$ for all vectors $c \in \{0,1\}^n$.

Theorem 31 *B is cross-perfect iff P is integral.*

Proof. Consider the line graph, $L(B)$, of B. For every 4-hole in $L(B)$, add the two cords. Let the new graph be called $L^{'}(B)$. We refer to $L^{'}(B)$ as the *modified* line graph of B. Then, $E^{'} \subseteq E$ is a cross-free matching in B iff $V(E^{'})$ is a independent set in $L^{'}(B)$. The clique polytope for $L^{'}$ is $Q = \{x : Sx \leq 1, x \in R_+^n\}$ where the rows of S correspond to maximal independent sets and the columns correspond to the nodes of $L^{'}$. But $Q = P$. Thus, B is cross-perfect implies that $w(H) = \gamma(H)$ for every node induced subgraph H of $L^{'}$ where $w(H)$ denotes the cardinality of the maximum clique in H and $\gamma(H)$ is the chromatic number of H. Then, $L^{'}$ is perfect by the weak perfect graph theorem [12][13]. It follows that Q and hence P is integral. Conversely, if P is integral, so is Q and hence $w(H) = \gamma(H)$ for every node induced subgraph H of $L^{'}$. Thus, $w^*(B_c) = \chi^*(B_c)$ for all vectors $c \in \{0,1\}^n$ and B is cross-perfect.

Note 1: An alternative proof of Theorem 3.1 was suggested by Prof. R. Chandrasekaran: The column-intersection graph of the matrix K^* is the complement graph of the modified line graph $L^{'}(G)$ defined above. The definition of the cross-perfectness and the perfect graph theorem [12] then imply that P is integral.
Note 2: Even though the proof of Theorem 3.1 uses the correspondence between the biclique number w^* and the clique number w, the structure of these two problems is fundamentally different. In fact, even the complexity question of computing w^* was open until the recent result of Peeters [15]. Theorem 3.1 and Theorem 3.4 (given below) are thus remarkable since they mention a duality,

similar to the classical duality in the perfect graph literature, for fundamentally different problems.

In general, the biclique polytope, P, is not integral. This follows because if P were integral, so would the clique polytope, Q, of the graph $L^{'}(B)$ which in turn would mean that $L^{'}(B)$ is perfect. However, it is easy to show a $L^{'}(B)$ which contains an odd hole of length 5 as a node-induced subgraph (see figure below).

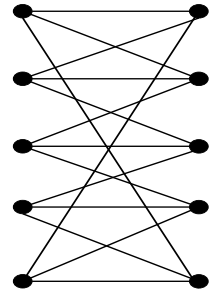


Fig. 1. A bipartite graph whose biclique polytope is not integral

Corollary 32 *B is cross-perfect iff the modified line graph $L^{'}(B)$ is perfect.*

This observation suggests that the property of cross-perfectness in bipartite graphs can be studied by studying perfectness in modified line graphs. If B is cross-perfect then, by Theorem 3.1, the constraints $K^*x \leq 1$ are sufficient to describe P_I. Note that every clique in $L^{'}$ does not correspond to a biclique in B. However, every *maximal* clique in $L^{'}$ does correspond to a biclique. Hence, a maximum clique, which is also a maximal clique, corresponds to a maximum biclique in B. This observation immediately gives us the following result.

Theorem 33 *If B is cross-perfect then $w^*(B)$ and $\chi^*(B)$ are polynomially solvable.*

Every independent set in $L^{'}$ corresponds to a cross-free matching in B. Consider a covering, C, of the nodes of $L^{'}$ by cliques. A clique c^* in C either corresponds to a biclique in B or if not, c^* is contained in a maximal clique which corresponds to a biclique in B. Thus, $\alpha(L^{'}) = m^*(B)$ and $\theta(L^{'}) = \beta^*(B)$. Let H be a node induced subgraph of $L^{'}$ and $c \in \{0,1\}^n$ be the incidence vector of the nodes corresponding to H. Then, it is easy to see that $\alpha(H) = m^*(B_c)$ and $\theta(H) = \beta^*(B_c)$. This correspondence along with the weak perfect graph theorem [12][13] gives us the following result.

Theorem 34 *The following are equivalent*

1. *$w^*(B_c) = \chi^*(B_c)$ for all vectors $c \in \{0,1\}^n$.*
2. *$m^*(B_c) = \beta^*(B_c)$ for all vectors $c \in \{0,1\}^n$.*

Theorem 35 *Node-induced subgraphs of cross-perfect graphs are cross-perfect*

Proof. The fractional biclique polytope, P_H, corresponding to a node induced subgraph H of B is a face of P. Hence if P is integral, so is P_H.

Corollary 36 *If B is cross-perfect then $w^*(H) = \chi^*(H)$ for all node induced subgraphs H of B.*

Getting an efficient recognition algorithm for cross-perfect bipartite graphs and the validity of the strong perfect graph conjecture [2][9] for modified line graphs (defined in Theorem 3.1) are outstanding open questions. Also, it is not known whether the class of modified line graphs of cross-perfect bipartite graphs is a new class of perfect graphs.

Acknowledgement

I thank Prof. R. Chandrasekaran for helpful discussions and suggestions.

References

1. Berge, C., Les problemes de colorations en theorie des graphes, *Publication of the Institute of Statistics, University of Paris,* 9, 123-160, 1960.
2. Berge, C., *Graphs and Hypergraphs*, North-Holland, 1973.
3. Chaty, C and Chein, M., Ordered matchings without alternating cycles in bipartite graphs, *Utilitas Mathematica,* 16, 183-187, 1979.
4. Cohen, B and Skiena, S., Optimizing combinatorial library construction via split synthesis, *Proc. Third International Conf. on Computational Molecular Biology (RECOMB 99)*, 1999. To appear in *Journal of Combinatorial Chemistry.*
5. Dawande, M., Keskinocak, P., Tayur, S., *On the Biclique Problem in Bipartite Graphs.* GSIA Working Paper 1996-04, Carnegie Mellon University, 1996.
6. Furka, A., Sebestyen, M., Asgedom, M. and Dibo, G., General method for rapid synthesis of multicomponent peptide mixtures, *Int. J. Peptide Protein Research,* 37, 487-493, 1991.
7. Ganter, B. and Wille, R., *Formale Begriffsanalyse – Mathematische Grundlagen.* Springer Verlag, 1996.
8. Garey, M. R. and Johnson, D. R., *Computers and Intractability*, W. H. Freeman and Co., San Francisco, 1979.
9. Golumbic, M.C., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, 1980.
10. Krogdahl, S., The dependence graph for bases in a matroid, *Discrete Mathematics,* 19, 47-59, 1977.
11. Lam, K., Salmon, S., Hersh, E., Hruby, V., Kazmierski, W. and Knapp, R., A new type of synthetic peptide library for identifying ligand-binding activity, *Nature,* 354, 82-86, 1991.
12. Lovasz, L., Normal hypergraphs and the perfect graph conjecture, *Discrete Mathematics,* 2, 253-267, 1972.
13. Meyniel, H., On the perfect graph conjecture, *Discrete Mathematics,* 16, 339-342.
14. Nemhauser, G.L. and Wolsey, L.A. (1988), *Integer Programming and Combinatorial Optimization,* Wiley, N.Y.
15. Peeters, R. *Personal Communication.*
16. Pulleyblank, W. R., Alternating cycle free matchings, *CORR Report 82-18,* Department of Combinatorics and Optimization, University of Waterloo, 1982.

Some Results on Orthogonal Factorizations*

Haodi Feng

Department of Computer Science, City University of Hong Kong, Hong Kong
fenghd@cs.cityu.edu.hk

Abstract. Consider a graph $G = (V, E)$ with an $[a, b]-$factorization $\mathcal{F} = \{F_1, F_2, ..., F_m\}$. It is proved in this paper that:

1. there is an $m-$matching of G to which $\mathcal{F}$ is orthogonal if $n = |V(G)| \geq (2 + \frac{b}{a})(m - 1)$ for $b \geq 2a$ and $n \geq 3.26m$ for $b = a$;
2. if $\sqrt{2b} \leq a \leq b$, then for any given edge e of G,there is a $[1, a]-$subgraph H of G such that e is included in H and $\mathcal{F}$ is orthogonal to H.

1 Introduction

All graphs considered are undirected graphs without loops and without multiple edges. Let G be a graph. we denote by $V(G)$ and $E(G)$ the vertex set and edge set of G respectively. Denote by $d_G(v)$ the degree of v in G. Let a and b be two integers. An $[a, b]-$factor of G is a spanning subgraph F of G such that $a \leq d_F(v) \leq b$ for every $v \in V(F)$. In particular, if $a = b = k$, then an $[a, b]-$factor is a $k-$factor. For simplicity and in case when there is no ambiguity, we denote a subgraph and its edge set with the same symbol. An $[a, b]-$factorization $\mathcal{F} = \{F_1, F_2, ..., F_m\}$ of G is a partition of $E(G)$ into edge disjoint $[a, b]-$factors $F_1, F_2, ..., F_m$. Similarly, we define $k-$factorization as a partition of $E(G)$ into $k-$factors. Let H be a subgraph of G with m edges and $\mathcal{F} = \{F_1, F_2, ..., F_m\}$ an $[a, b]-$factorization of G, then $\mathcal{F}$ is called orthogonal to H if each $F_i, 1 \leq i \leq m$ has exactly one edge in common with H. Other definitions and terminologies can be found in [2].

Alspach B. posed the following conjecture:

Conjecture 1. [1] Let G be a $2d-$regular graph, $\mathcal{F} = \{F_1, F_2, ..., F_m\}$ be a $2-$factorization of G, then there exists a $d-$matching M of G to which $\mathcal{F}$ is orthogonal.

G.Z.Liu et al. [4] proved that the conjecture is true when $|V(G)| \geq 4d - 5$ and extend the question to $k-$factorizations and $[1, k]-$subgraphs. Later, M. Kouider et al. [3] improved the result of [4] to $|V(G)| \geq 3.23d$.

In this paper, we improve the above results and extend them to $[a, b]-$factorization. The main techniques of this paper come from those in [4] and [3].

* Research is partially supported by a grant from the Research Grants Council of Hong Kong SAR (CityU 1074/00E) and a grant from CityU of Hong Kong (Project No.7001215).

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 414–419, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Let A and B be two disjoint subsets of $V(G)$, then for any $1 \leq i \leq m$, denote by $E_i(A,B)$ the set of edges of factor F_i with one end in A and the other in B, by $E_i(A)$ the set of edges of factor F_i with both ends in A. Let $k_1, ..., k_j$ be j different indexes between 1 and m, then denote by $E_{k_1,...,k_j}(A,B)$ the set of edges of all the factors $F_{k_1}, ..., F_{k_j}$ with one end in A and the other in B.

2 Main Theorems and Proofs

Theorem 1. *Let a, b and m be positive integers with $a \leq b$. Let G be a graph with an $[a,b]-$factorization $\mathcal{F} = \{F_1, F_2, ..., F_m\}$. Then there is an $m-$matching of G to which $\mathcal{F}$ is orthogonal if $n = |V(G)| \geq (2 + \frac{b}{a})(m-1)$ for $b \geq 2a$ and $n \geq 3.26m$ for $b = a$.*

Proof. We apply induction on m. When $m = 1$, the theorem is obviously true. Suppose the theorem is true for $m-1$. We will prove that the theorem holds for $m \geq 2$. By induction hypothesis, there is an $(m-1)-$matching M' of G to which $\mathcal{F}' = \{F_1, F_2, ..., F_{m-1}\}$ is orthogonal. Assume that there is no $m-$matching of G as described in theorem 1. We will see a contradiction.

W.l.g., suppose $M' = \{e_1, e_2, .., e_{m-1}\}$, where $e_i = x_i y_i \in F_i, 1 \leq i \leq m-1$. Let $A = \{x_i, y_i : 1 \leq i \leq m-1\}$, $B = V(G) \setminus A$. Then $|A| = 2(m-1)$, $|B| = n - 2(m-1)$. Since there is no $m-$matching of G satisfying the conclusion of theorem 1 according to our assumption, we have that $E_m(B) = \phi$. Then $E_m = E_m(A,B) + E_m(A)$.

We now reindex the edges in M' and let s denote the maximum index subject to the following two conditions,

1. there exist two distinct vertices x'_1 and y'_1 in B satisfying that $x_1x'_1$ and $y_1y'_1$ are edges in F_m;
2. for each index i between 2 and s, there exist two distinct vertices x'_i and y'_i satisfying that $x_ix'_i$ and $y_iy'_i$ are edges in F_{i-1}. All the $\{x'_i, y'_i\}(i \geq 1)$ are disjoint.

Consider the subset $S = \{e_1, ..., e_s\}$ of M'. We claim that S is nonempty. For there are always some edges, say $e_j = x_jy_j$ in M' which are adjacent to two independent edges in $E_m(A,B)$. Otherwise, for each edge $e_j = x_jy_j \in M'$, either the edges in $F_m(A,B)$ are only incident with one end point of e_j or both the edges in $F_m(A,B)$ incident with x_j and y_j end in the same vertex in B. Therefore, we always have $a|B| \leq b|M'| = b\frac{|A|}{2}$, and subsequently we have

$$n = |A| + |B| \leq (1 + \frac{b}{2a})|A| = (2 + \frac{b}{a})(m-1), \tag{1}$$

a contradiction to our assumption.

For each index k between 1 and s, denote by A_k the set of the ends of $e_1, e_2, ..., e_k$, by A'_k the complement of A_k in A; denote by B_k the set of $x'_i, y'_i, 1 \leq i \leq k$, by B'_k the complement of B_k in B. Then we have:

$$|A_k| = |B_k| = 2k, |A'_k| = 2(m-1-k), |B'_k| = n - 2m + 2 - 2k. \tag{2}$$

Since according to our assumption there is no matching of G to which $\mathcal{F}$ is orthogonal, we have

$$E_S(A_S, B'_S) = E_S(B_S, B'_S) = E_S(B'_S,\ B'_S) = \emptyset. \tag{3}$$

Besides, due to the maximality of s, similar as the discussion before, any edge x with $x \in E_{M'}(A'_S)$ is adjacent to at most one independent edge in $E_S(A'_S, B'_S)$. Therefore we have:

$$|B'_S| \leq \frac{b}{2a}|A'_s|, \tag{4}$$

which implies that:

$$n = |B'_s| + 2m - 2 + 2s \leq \frac{b}{2a}|A'_s| + 2m - 2 + 2s = (2 + \frac{b}{a})(m-1) + (2 - \frac{b}{a})s. \tag{5}$$

Similarly, we have:

$$\begin{array}{l} E_{k,k+1,\ldots,s}(A_k, B'_k) = \emptyset, \\ E_{k,k+1,\ldots,s}(B_k, B'_k) = \emptyset, \\ E_{k,k+1,\ldots,s}(B'_k, B'_k) \subseteq \{x'_i y'_i | k+1 \leq i \leq s\}, \\ E_{k,k+1,\ldots,s}(A_k \cup B_k, A_k \cup B_k), \\ \quad \subseteq \{x_i y'_i, y_i x'_i, x'_i y'_i | 1 \leq i \leq k\} \cup \{e_k\}, \end{array} \tag{6}$$

Subsequently we have:

$$\begin{array}{l} |E_{k,k+1,\ldots,s}(A_k, A'_k)| \geq a|A_k|(s-k+1) - |A_k| - 2, \\ |E_{k,k+1,\ldots,s}(B_k, A'_k)| \geq a|B_k|(s-k+1) - 2|B_k|, \\ |E_{k,k+1,\ldots,s}(B'_k, A'_k)| \geq a|B'_k|(s-k+1) - 2(s-k). \end{array} \tag{7}$$

And therefore,

$$\begin{array}{rl} |E_{k,k+1,\ldots,s}(A'_k, A_k \cup B)| \geq & a(s-k+1)(|A_k| + |B|) \\ & -(6k+2) - 2(s-k). \end{array} \tag{8}$$

On the other hand, counting the edges in $E_{k,k+1,\ldots s}(A'_k, A_k \cup B)$, we have that:

$$|E_{k,k+1,\ldots,s}(A'_k, A_k \cup B)| \leq b|A'_k|(s-k+1) - 2(s-k). \tag{9}$$

Subsequently we have,

$$\begin{array}{rl} b|A'_k|(s-k+1) - 2(s-k) \geq & a(s-k+1)(|A_k| + |B|) \\ & -(6k+2) - 2(s-k). \end{array} \tag{10}$$

Therefore,

$$a(s-k+1)(2k+n-2m+2) - b(s-k+1)(2m-2-2k) \leq 6k+2. \tag{11}$$

Hence, we have:

$$n \leq \frac{2m(a+b)}{a} - \frac{2(a+b)}{a} - \frac{2k(a+b)}{a} + \frac{6k+2}{a(s-k+1)}. \tag{12}$$

Now we have two cases to discuss.

Case 1. $b \geq 2a$.

From (5) we have that

$$n \leq (2 + \frac{b}{a})(m-1), \tag{13}$$

a contradiction to our assumption.

Case 2. $b = a$.

From (5), we have $n \leq 3m - 3 + s$. Now, if $s \leq 3$, then $n \leq 3m < 3.26m$, a contradiction; if $s = 4$, then $m \geq s + 1 = 5$ and $n \leq 3m + 1 < 3.26m$, still a contradiction. Therefore, in the following discussion, we assume that $s \geq 5$.

Set $k = s - 4$ in equation (12), then we have :

$$n \leq \frac{2m(a+b)}{a} - \frac{2(a+b)}{a} - \frac{2k(a+b)}{a} + \frac{6(s-4)+2}{5a}, \tag{14}$$

which is equivalent to the following equation:

$$5n \leq (10 + \frac{10b}{a})m - (10 + \frac{10b}{a} - \frac{6}{a})s + 30 - \frac{22}{a} + \frac{30b}{a}. \tag{15}$$

From (5), we have $s \geq n - 3m + 3$ and subsequently we have

$$5n \leq (10 + \frac{10b}{a})m - (10 + \frac{10b}{a} - \frac{6}{a})(n - 3m + 3) + 30 - \frac{22}{a} + \frac{30b}{a}. \tag{16}$$

Therefore,

$$n \leq \frac{40a + 40b - 18}{15a + 10b - 6}m - \frac{4}{a} < \frac{80a - 18}{25a - 6}m \leq 3.26m, \tag{17}$$

a contradiction to our assumption that $n \geq 3.26m$.

Thus completes the proof of Theorem 1. □

Corollary 1. *Let G be a $kd-$graph with vertex number $n \geq 3.26d$ and let $\mathcal{F} = \{F_1, F_2, ..., F_d\}$ be a $k-$factorization of G. Then G has a $d-$matching to which $\mathcal{F}$ is orthogonal.*

Proof. We can prove this corollary by setting $a = b = k$ in Theorem 1. □

Corollary 1 greatly improved G.Z.Liu et al. 's results concerning orthogonal $k-$factorizations in [4].

Theorem 2. *Let a, b and l be integers such that $\sqrt{2b} \leq a \leq b$. Let G be a graph, $\mathcal{F} = \{F_1, F_2, ..., F_l\}$ be an $[a,b]-$factorization of G. Then for every given edge e of G, G has a $[1,a]-$subgraph H such that $e \in H$ and to H $\mathcal{F}$ is orthogonal.*

Proof. We use induction on l. When $l = 1$, the theorem is trivial. Suppose the theorem is true for $l \leq m-1$, we will prove it is true for $l = m$.

W.l.g., suppose $e \notin F_m$. By induction assumption, G has a $[1, a]-$subgraph H', such that $e \in H'$ and that to H', $\mathcal{F}' = \{F_1, F_2, ..., F_{m-1}\}$ is orthogonal. Suppose G has no satisfying $[1, a]-$subgraphs, we will see a contradiction to our given conditions of theorem 2.

Let

$$S = \{x_i, y_i : x_i y_i \in E(H')\}, T = V(G) \setminus S. \tag{18}$$

Let

$$\begin{aligned} S_l &= \{x \in S \text{ and } d_{H'}(x) < a\}, \\ S_a &= \{x \in S \text{ and } d_{H'}(x) = a\}. \end{aligned} \tag{19}$$

Since G has no satisfying $[1, a]-$subgraphs, we have that $E_m = E_m(S_a, V(G))$. On the other hand, $E_m(T, S_l) = \emptyset$, $E_m(T, T) = \emptyset$. Therefore,

$$E_m(T, V(G)) = E_m(T, S_a), \tag{20}$$

and subsequently,

$$|E_m(T, V(G))| \leq |E_m| = |E_m(S_a, V(G))|. \tag{21}$$

Notice that

$$|E_m(T, V(G))| \geq a|T|, \ |E_m(S_a, V(G))| \leq b|S_a|, \tag{22}$$

we have

$$a|T| \leq |E_m(T, V(G))| \leq |E_m(S_a, V(G))| \leq b|S_a|, \tag{23}$$

and subsequently we have

$$|T| \leq \frac{b}{a}|S_a|. \tag{24}$$

Now we have two cases to discuss:

Case 3. $a = 2$.

In this case, H' is a $[1, 2]-$subgraph of G. Hence,

$$2(m-1) = |S_l| + 2(|S| - |S_l|). \tag{25}$$

Then we have

$$|S_l| = 2(|S| - m) + 2, \ |S_a| = |S| - |S_l| = 2m - |S| - 2. \tag{26}$$

Notice that

$$|T| = n - |S| \leq \frac{b}{a}|S_a| = \frac{b}{a}(2m - |S| - 2), \tag{27}$$

we have

$$n \leq \frac{2b}{a}m - \frac{2b}{a} - (\frac{b}{a})|S| < \frac{2b}{a}m. \tag{28}$$

On the other hand, since $n \geq ma + 1$, we have

$$ma + 1 < \frac{2b}{a}m, \tag{29}$$

which implies $a < \sqrt{2b}$, a contradiction to $a \geq \sqrt{2b}$.

Case 4. $a \geq 3$.

Since H' has $m-1$ edges, we have $a|S_a| \leq 2(m-1)$ and therefore $|S_a| \leq \frac{2(m-1)}{a}$. Recall that $|T| \leq \frac{b}{a}|S_a|$, $|T| = n - |S|$, we have

$$n \leq |S| + \frac{b}{a}|S_a| \leq 2m - 2 + \frac{2b(m-1)}{a^2} = (2 + \frac{2b}{a^2})m - 2 - \frac{2b}{a^2}. \tag{30}$$

Since $a \geq \sqrt{2b}$, we have $\frac{2b}{a^2} \leq 1$ and therefore $n \leq 3m - 2 < ma + 1$, a contradiction to $n \geq ma + 1$.

Theorem 2 is proved. □

Remark 1. We can derive the results concerning $[1,k]-subgraph$ in [4] by setting $a = b = k$ in Theorem 2.

Acknowledgment

The author would like to thank Dr. Xiaotie Deng, Dr. Guojun Li, Prof. Guizhen Liu and all the referees for their helpful comments and advice on this paper.

References

1. B. Alspach: Problem 89. Discrete Math. **69** (1988) 106
2. J. A. Bondy, U. S. R. Murty: Graph Theory with Applications. Elsevier-North Holland, New York (1976)
3. M. Kouider, D. Sotteau: On the Existence of a Matching Orthogonal to a 2−Factorization. Discrete Math. **73** (1989) 301–304
4. G. Z. Liu, B. Alspach, K. Heinrich: Some Results on Orthogonal Factorizations. Advances in Mathematics(in China) **21** (2) (1992) 211–215.

Cluttered Orderings for the Complete Graph

Myra B. Cohen[1], Charles J. Colbourn[2], and Dalibor Froncek[3]

[1] Computer Science, University of Auckland, Auckland, New Zealand
myra@cs.auckland.ac.nz
[2] Computer Science, University of Vermont, Burlington, VT 05405, U.S.A.
colbourn@cs.uvm.edu
[3] Applied Mathematics, Technical Univ. Ostrava, 708 33 Ostrava, Czech Republic
dalibor.froncek@vsb.cz

Abstract. In a systematic erasure code for the correction of two simultaneous erasures, each information symbol must have two associated parity symbols. When implemented in a redundant array of independent disks (RAID), performance requirements on the update penalty necessitate that each information symbol be associated with no more parity symbols than the two required. This leads to a simple graph model of the erasure codes, with parity symbols as vertices and information symbols as edges. Ordering the edges so that no more than f check disks (vertices) appear among any set of d consecutive edges is found to optimize access performance of the disk array when d is maximized. These *cluttered orderings* are examined for the complete graph K_n. The maximum number d of edges is determined precisely when $f \leq 5$ and when $f = n - 1$, and bounds are derived in the remaining cases.

1 Background

Let $G = (V, E)$ be a graph on $n = |V|$ vertices and $m = |E|$ edges. Let $d \leq m$ be a positive integer, called the *window*. Let $E = \{e_0, e_1, \ldots, e_{m-1}\}$. An *edge ordering* of G is a permutation π of the edge indices $\{0, 1, \ldots, m-1\}$. For the graph G with edge ordering π, and window d, define the $m + 1 - d$ graphs $\{G_i^{\pi,d} : 0 \leq i \leq m - d\}$ by setting $G_i^{\pi,d}$ to be the graph containing edges $\{e_{\pi(i)}, e_{\pi(i+1)}, \ldots, e_{\pi(i+d-1)}\}$. Then, when sets of d edges that are consecutive under the ordering π are accessed, the graphs $\{G_i^{\pi,d}\}$ represent the possible subgraphs accessed.

In the application to be described, the cost of accessing a subgraph of d consecutive edges is measured by the sum of the number of edges and the number of vertices of nonzero degree in the subgraph. Since each has d edges, any reduction in access cost results from varying the numbers of vertices. Hence we define $n_i^{\pi,d}$ to be the number of vertices of nonzero degree in $G_i^{\pi,d}$. The (average) *d-access cost* of graph G under ordering π is defined to be $\frac{\left(\sum_{i=0}^{m-d} n_i^{\pi,d}\right)}{(m+1-d)}$. The *$d$-maximum access cost* of graph G under ordering π is defined to be $\max_{i=0}^{m-d} n_i^{\pi,d}$. Maximum access cost and average access cost measure different aspects of performance,

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 420–431, 2001.
© Springer-Verlag Berlin Heidelberg 2001

but evidently address similar goals. An ordering is (d, f)-*cluttered* if it has d-maximum access cost equal to f. Let $m_d(G)$ denote the minimum value of f, over all edge orderings π of G, of the d-maximum access cost. It may happen that $m_d(G) = m_{d+1}(G)$ (see [7], for example). We write $c_f(G)$ for the maximum value of d for which a (d, f)-cluttered ordering of G exists. When $G = K_n$, we use the notation $m_d(n)$ and $c_f(n)$.

In this paper, we develop two combinatorial constructions for cluttered orderings of the complete graph K_n. The first is for small windows (§3) and the second for large windows (§4). Each is limited to specific parameter selections for d, f, and n, and hence may not provide the required solution for parameters arising in a particular application. However, we derive a powerful extension of the combinatorial labelling technique for small windows which dramatically increases its application. In section 5, we also develop an inflation technique that permits us to obtain orderings for specific parameter situations from solutions for smaller cases.

2 An Application to RAID

Secondary storage devices play an important role in system performance. Improvements in magnetic disk speed have not kept pace with improvements in CPU performance. Current systems therefore have the potential to be I/O bound. This, coupled with the requirement for high reliability and availability, has resulted in use of redundant arrays of independent disks (RAID); see [3,9,11,12,13,14]. RAID maps larger disk reads or writes to multiple smaller reads and writes. It splits (*stripes*) these across several disks in parallel along a *parity stripe* [16]. Redundant information is maintained in order to reconstruct the array in the case of disk failure. Parity computations are employed to retain the redundant information.

A catastrophic failure renders a disk unreadable; this is an *erasure*. A code that corrects for t erasures is a t-*erasure code*. One important metric in erasure codes for disk array architecture is the *update penalty*. This is the number of disks whose contents must be changed each time a write occurs on a single disk in the array. In a t-erasure code, the minimal update penalty is t.

In the schemes that we examine, there are $c + k$ disks. Within each stripe across these disks, k positions are reserved for information while c express parity functions of the information. The code used is *systematic*, because information is expressed in an unencoded format. The equations for an erasure code using parity for redundancy in disk arrays can be represented as a a *parity check matrix*, $H = [P|I]$. This is a $c \times (k + c)$ matrix where the columns of P are indexed by the information disks, the columns of I are indexed by check disks and the rows of P and I by the check disks. I is the $c \times c$ identity matrix. A disk failure is *recoverable* if and only if the failed columns are linearly independent modulo 2 [10].

In this setting, for each write to an information disk, we must also update each of its check disks. Writes are expensive operations in a disk array. Update penalty

is a useful metric, but it fails to describe the observed write penalty. Consider writing information to the array. One method is to write to each information disk to be updated; then each of the information disks in their equations is read, in order to calculate the new parity; and lastly the changed parity is written to each of the check disks. This is unnecessarily expensive when writing to a small fraction of disks in an array. For large disk arrays with multiple erasure protection, when the number of disks to be written is relatively small compared with the number of information disks, all writes are implemented as a *read-modify-write* (see [6,11]). To write to a disk, we only need to calculate the change to its check disks, before updating them. Therefore we read all of the check and information disks involved. Then the change to the parity is computed, the check disks are written and the new data is written. For a write involving only one information disk in a 2-erasure code, this requires six I/O accesses. The actual penalty for an update penalty of t is $2t + n$ accesses where n is the number of information disks. For small writes, this is less expensive than reading the entire stripe.

Our previous work on 3-erasures shows that the ordering of the parity check matrix may reduce the observed update penalty when writing across three disks at a time [5,6]. Since information disks share check disks, we can improve the observed penalty if we impose an ordering and assume some intermediate buffering or parity logging (see [5,17]). If data is striped sequentially along disks in a parity check matrix and these disks share check disks, the actual number of reads and writes can be reduced.

The connection to cluttered edge orderings is as follows. Columns of the parity check matrix contain two '1' entries, and hence if we interpret check disks as vertices, every information disk is associated with an unordered pair of check disks, i.e. with an edge. The parity check matrix therefore represents a graph; indeed if it does correct two erasures, the graph has no multiple edges. Therefore the maximum number of information disks that can be employed corresponds to the selection of the complete graph to form the parity check matrix. If we label the first h columns of the parity check matrix with h distinct pairs on c vertices and then label the identity portion of the matrix with $h + 1$ through $h + c$ we form a parity check matrix. If we then order the h columns using a cluttered ordering of the corresponding graph, and stripe data sequentially, this minimizes the maximum number of check disks affected by a write operation.

3 Wrapped Labellings

In this section, we develop a technique for producing specific (d, f)-cluttered orderings of K_n when f is relatively small. We define a graph $G(h, w)$ on $h(w+1)$ vertices. The vertex set is $V = \{v_{ij} : 0 \le i < h, 0 \le j \le w\}$. The edge set is $E = \{\{v_{ij}, v_{kj}\} : 0 \le i < k < h, 0 \le j < w\} \cup \{\{v_{ij}, v_{k,j+1}\} : 0 \le k \le i < h, 0 \le j < w\}$. There are $w \cdot h^2$ edges; indeed, $G(h, w)$ has a natural edge-partition into w copies of $G(h, 1)$. A *ρ-labelling* of a graph $G = (V, E)$ is a mapping $\rho : V \mapsto \mathbb{Z}_{2|E|+1}$ for which $\{\pm(\rho(r) - \rho(s)) : \{r, s\} \in E\}$ contains all nonzero

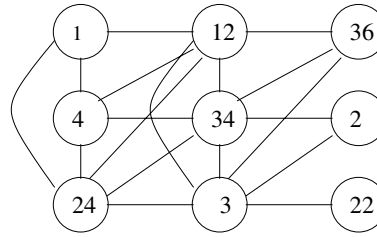


Fig. 1. Wrapped ρ-labelling of $G(3,2)$

numbers in $\mathbb{Z}_{2|E|+1}$; see [15]. These ρ-labellings have been extensively studied as methods of partitioning (or *decomposing*) the edge set of the complete graph $K_{2|E|+1}$ into copies of G [1,15]. Under suitable conditions, they can also be used to constrain the graphs induced among all sets of $|E|$ consecutive edges; a simple example is given in [7]. We pursue a more general formulation next.

3.1 Wrapped Labelling of $G(h, w)$

A ρ-labelling of $G(h,w)$ is a *wrapped ρ-labelling* if it satisfies $\{\alpha + \rho(v_{i0}) : 0 \leq i < h\} = \{\rho(v_{iw}) : 0 \leq i < h\}$ for some α which is relatively prime to $2|E|+1$. Figure 1 shows a wrapped ρ-labelling for $G(3,2)$; here $\alpha = -2$.

Proposition 1. *If $G(h,w)$ has a wrapped ρ-labelling, then there is an ordering of the edges of K_{2wh^2+1} in which every set of h^2 consecutive edges induces a subgraph on at most $2h$ vertices, i.e. a $(h^2, 2h)$-cluttered ordering. Thus $m_{h^2}(2wh^2+1) \leq 2h$ and $c_{2h}(2wh^2+1) \geq h^2$.*

Proof. Let $\alpha + \rho$ be the labelling obtained by adding α to each label assigned by ρ, reducing modulo $2wh^2+1$. Starting with (G_{hw}, ρ), we can identify the vertices $\{v_{iw}\}$ with the vertices $\{v_{i0}\}$ of $(G_{hw}, \alpha+\rho)$ so that identified vertices have the same label (since the labelling is a wrapped ρ-labelling). Repeating this addition of α, we encounter every translate of the labelling ρ since α and $2wh^2+1$ are relatively prime. Indeed, in the process, every edge of K_{2wh^2+1} arises in exactly one of the translated graphs constructed, since every edge difference arises exactly once in ρ. An example, adjoining the first four translates of $G(3,1)$ with wrapped ρ-labellings ρ, $1+\rho$, $2+\rho$, and $3+\rho$, is shown in Figure 2.

Our task is now to order the edges of $G(h,1)$. We first list the $\binom{h}{2}$ edges on $\{v_{i0}\}$ in any order. Then we start with $\{v_{00}, v_{01}\}$; $\{v_{10}, v_{01}\}$, $\{v_{10}, v_{11}\}$; and in general, $\{v_{i0}, v_{01}\}, \ldots, \{v_{i0}, v_{i1}\}$. In other words, we list the edges of the form $\{v_{i0}, v_{j1}\}$ so that $\{v_{i0}, v_{j1}\}$ precedes $\{v_{k0}, v_{\ell 1}\}$ if and only if $i < k$, or $i = k$ and $j < \ell$. If, within every copy of $G(h,1)$, edges are ordered in this manner, then we claim that every set of h^2 consecutive edges induces at most $2h$ vertices. Consider sliding a window of size h^2 through the first and second copies of $G(h,1)$. Initially there are $2h$ vertices. The first $\binom{h}{2}$ transitions move the edges on $\{v_{i0}\}$ to the corresponding edges on $\{v_{i1}\}$; no new vertices are encountered.

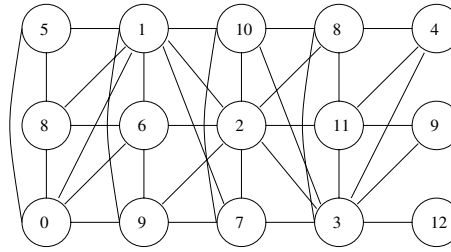


Fig. 2. First four translates of $G(3, 1)$

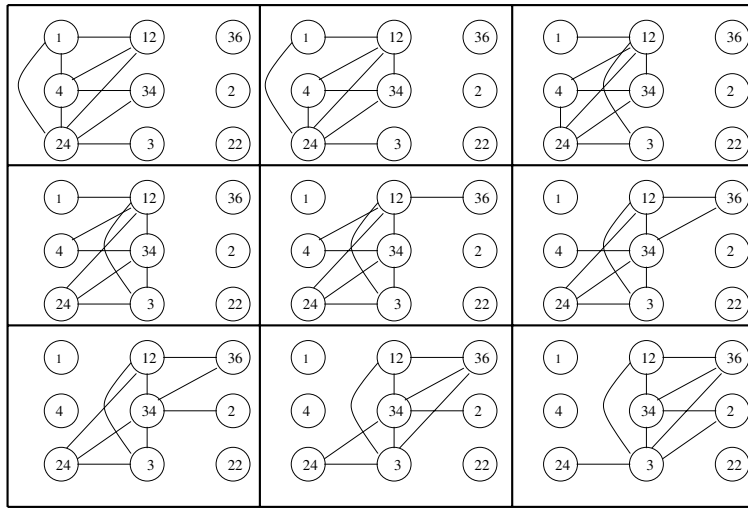


Fig. 3. First nine windows for $G(3, 2)$

The next transition drops the only edge still incident at v_{00} and adds an edge to v_{02}. The next edge introduced involves v_{02}, so no new vertex is seen. Then the last edge incident at v_{10} is dropped, and an edge to v_{12} added. Carrying this on, whenever a new vertex is to be introduced, say v_{i2}, the last remaining edge incident with v_{i0} is being dropped. Thus after h^2 transitions, the window contains the second copy of $G(h, 1)$ in the $G(h, w)$. See Figure 3 for an illustration of the first nine windows (arranged horizontally) of the ordering of $G(3, 2)$. Transitions into all copies of $G(h, 1)$ are effected in this way. When the first $G(h, w)$ is fully seen, since the second $G(h, w)$ is attached to it as described above, we can simply continue with the same argument. Thus every set of h^2 consecutive edges induces at most $2h$ vertices. □

Enlarging the window provides a further useful result. Consider $t \cdot h^2 + \binom{x+1}{2}$ consecutive edges in the ordering. The argument in the proof establishes that at most $(t + 1)h + x$ vertices are involved in these edges, and hence:

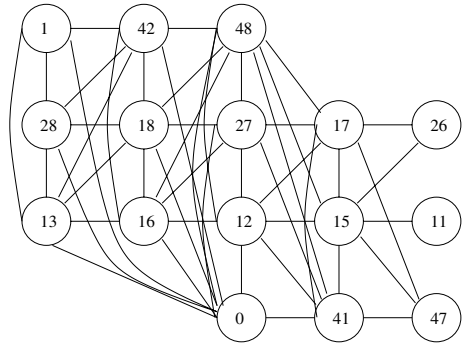


Fig. 4. $\Gamma(3,2)$ and its first translate

Proposition 2. *If $G(h,w)$ has a wrapped ρ-labelling, then $c_{(t+1)h+x}(2wh^2+1) \geq th^2 + \binom{x+1}{2}$ for all $t \geq 1$.*

We show some $h \times (w+1)$ boxes giving wrapped ρ-labellings (found by a backtracking algorithm) for $G(h,w)$:

3 1 0 4	5 1 8 6 0 9	1 12 36 4 34 2 24 3 22	1 13 3 54 2 44 25 0 4 52 20 2	1 8 13 58 72 2 21 70 32 0 4 52 43 22 2	1 8 57 17 47 90 2 13 88 29 62 0 4 21 67 81 25 2	1 8 31 2 41 72 108 2 13 53 97 60 25 0 4 21 81 26 93 45 2

1 8 31 93 45 7 87 126 2 13 1 74 100 23 60 0 4 21 53 17 76 101 45 2	5 1 18 6 8 19 0 9	1 15 64 3 46 1 39 14 37 11 20 9	1 58 34 96 2 66 59 0 11 32 15 9 86 63 45 84	12 1 46 13 22 47 15 23 0 16

3.2 Wrapped ρ-Labelling of $\Gamma(h,w)$

A very similar approach enables us to treat windows which induce an odd number of vertices. We form a graph $\Gamma(h,w)$ which contains all of the vertices and edges of $G(h,w)$, along with one further vertex ∞, and further edges $\{\{\infty, v_{ij}\} : 0 \leq i < h, 0 \leq j < w\}$. $\Gamma(h,w)$ has $wh(h+1)$ edges. A *wrapped ρ-labelling* of $\Gamma(h,w)$ is a labelling for which $\{\alpha+\rho(\infty)\} \cup \{\alpha+\rho(v_{i0}) : 0 \leq i < h\} = \{\rho(\infty)\} \cup \{\rho(v_{iw}) : 0 \leq i < h\}$ for some α relatively prime to $2wh(h+1)+1$.

The edge ordering employed for $G(h,w)$ is extended to $\Gamma(h,w)$ by placing all edges involving ∞ at the beginning. The property of being wrapped ensures that the vertices $\{\infty\} \cup \{v_{i0} : 0 \leq i < h\}$ have the same labels in $(\Gamma(h,w),\rho)$ as those of $\{\infty\} \cup \{v_{iw} : 0 \leq i < h\}$ in $(\Gamma(h,w),\alpha+\rho)$, so that an identification of these vertices can be made as for wrapped ρ-labellings. Figure 4 shows a wrapped ρ-labelling of $\Gamma(3,2)$, adjoined to the first translate with $\alpha = -1$. The vertex labeled 0 is ∞ in the first $\Gamma(3,2)$, while the vertex labeled 48 is ∞ in the second.

Proposition 3. *If $\Gamma(h,w)$ has a wrapped ρ-labelling, then there is an ordering of the edges of $K_{2wh(h+1)+1}$ in which every set of $h(h+1)$ consecutive edges induces a subgraph on at most $2h+1$ vertices, i.e. a $(h(h+1), 2h+1)$-cluttered ordering. Thus $m_{h(h+1)}(2wh(h+1)+1) \leq 2h+1$ and $c_{2h+1}(2wh(h+1)+1) \geq h(h+1)$.*

This extends in the same manner as Proposition 2, to establish:

Proposition 4. *If $\Gamma(h, w)$ has a wrapped ρ-labelling, then $c_{(t+1)h+x+1}(2wh(h+1)+1) \geq th(h+1)+\binom{x+1}{2}$ for all $t \geq 1$.*

In the wrapped ρ-labellings to follow, we always take $\rho(\infty) = 0$.

4 12 1 3	1 15 24 7 2 6	1 26 16 36 3 7 31 2	1 8 35 16 48 3 31 25 36 2	1 7 48 10 18 60 3 12 26 50 33 2	1 7 30 46 36 14 72 3 12 54 25 53 40 2	1 7 20 64 53 10 29 84 3 12 37 14 38 69 51 2

1 7 20 53 36 78 24 39 96 3 12 30 81 22 49 86 65 2	1 7 20 41 93 56 29 54 77 108 3 12 30 74 34 95 71 26 62 2	1 7 20 41 79 19 74 40 88 53 120 3 12 30 14 50 89 17 62 25 77 2

1 7 20 41 56 114 70 53 96 28 73 132 3 12 30 14 78 31 110 29 61 99 48 2	11 24 7 10 1 6	1 42 48 28 18 27 13 16 12	1 30 66 72 3 52 12 2 35 4 57 34	1 61 70 13 96 3 81 8 46 2 7 49 80 28 6

31 40 27 30 7 26 1 6	33 60 23 32 19 22 7 18 1 6	89 112 85 88 73 84 67 72 53 66 9 52 1 8	113 144 69 112 65 68 59 64 51 58 39 50 23 38 1 22

Cohen and Colbourn [7] treated a restricted version of this problem, namely finding wrapped ρ-labellings of $\Gamma(1, w)$. They settled this problem completely in the affirmative, and established more. When $d = 3$, the access cost for a specific subgraph is at least 3, and at most 6. When three consecutive edges induce a triangle, we have access cost 3 and this is the only situation accounting for this minimum. However, when $\{e_i, e_{i+1}, e_{i+2}\}$ induces a triangle, $\{e_{i-1}, e_i, e_{i+1}\}$ cannot form a triangle unless $n = 3$, for $e_{i-1} = e_{i+2}$ is necessary if both are triangles. Now when three consecutive edges do *not* form a triangle, the fewest vertices induced is four, which can be realized by a path or a star on three edges. Hence the minimum 3-access cost of K_n is at least 3.5 when $\binom{n}{2} \equiv 0 \pmod 2$, and is at least $3.5 - \frac{1}{n^2-n-4}$ when $\binom{n}{2} \equiv 1 \pmod 2$.

An edge ordering for K_n which realizes this minimum when $d = 3$ is then sought. The specific question addressed is: When can the $\binom{n}{2}$ edges of the complete graph K_n be ordered by a permutation π, so that among the $\binom{n}{2} - 2$ subgraphs $\{G_i^{\pi,3}\}$, at least $\frac{1}{4}(n^2 - n - 6)$ subgraphs form triangles? Such an ordering of the edges of K_n is a *ladder ordering of pairs*. When $\binom{n}{2}$ is even, this requires that for any three consecutive edges e_i, e_{i+1}, e_{i+2} with $0 \leq i \leq \binom{n}{2} - 3$, the subgraph induced by these three edges contains three vertices when i is even, and four vertices when i is odd. A wrapped ρ-labelling of $G(1, w)$ gives a ladder ordering of pairs for $n = 4w + 1$. Cohen and Colbourn [7] prove:

Theorem 1. *A ladder ordering of pairs for K_n exists except possibly when $n \in \{15, 18, 22\}$.*

This has an immediate application to the determination of maximum access cost:

Proposition 5. *There is an edge ordering of K_n with $(2f-4)$-maximum access cost less than or equal to f. Hence, with the stated exceptions for n, the largest d for which $m_d(n) = f$ is at least $2f - 4$, i.e. $c_f(n) \geq 2f - 4$.*

Proof. A ladder ordering of K_n employs at most f vertices among any $2f-4$ consecutive edges, for all f. It remains only to treat the case when $f=3$, which is handled in [7]. □

We next develop a simple lower bound for $c_f(n)$.

Proposition 6. *For $f \geq 3$, $c_f(n) \leq \binom{f-1}{2} + 1$. Hence $c_3(n) = 2$ and $c_4(n) = 4$.*

Proof. Consider a (d, f)-cluttered ordering π on $G = K_n$. It must happen that for some value of i, $G_i^{\pi,d}$ and $G_{i+1}^{\pi,d}$ are on different sets of f vertices. But $G_i^{\pi,d}$ has at most one edge that does not appear in $G_{i+1}^{\pi,d}$, and hence contains a vertex of degree one. It follows that $G_i^{\pi,d}$ can contain at most $\binom{f-1}{2} + 1$ edges. □

This can be refined to prove that $c_5(n) = 6$, as follows. If an ordering π of K_n had 7-maximum access cost equal to five, then by the argument given, we can assume that $G_i^{\pi,d}$ and $G_{i+1}^{\pi,d}$ share a K_4. Upon considering the vertices of $G_{i+r}^{\pi,d}$ for $r = 2, 3, 4$, since no new vertices can be introduced, we find that the three edges $e_{\pi(i+1)}$, $e_{\pi(i+2)}$ and $e_{\pi(i+3)}$ share a common vertex which does not appear in $G_{i+4}^{\pi,d}$ (otherwise, the vertex y added in forming $G_{i+1}^{\pi,d}$ has become saturated and no more edges can be placed).

A more involved argument establishes that $c_6(n) \leq 9$, but a general bound is elusive.

3.3 A Generalization

There is no wrapped ρ-labelling of $G(h, 1)$ for $h \in \{6, 7, 8, 9\}$. However, removing the edge $\{v_{50}, v_{51}\}$ from $G_{6,1}$, we can find a wrapped ρ-labelling of the resulting graph, with $\rho(v_{00}) = \rho(v_{11}) - 1 = 27$, $\rho(v_{10}) = \rho(v_{21}) - 1 = 18$, $\rho(v_{20}) = \rho(v_{31}) - 1 = 41$, $\rho(v_{30}) = \rho(v_{41}) - 1 = 7$, $\rho(v_{40}) = \rho(v_{51}) - 1 = 3$, and $\rho(v_{50}) = \rho(v_{01}) - 1 = 0$. By the reasoning of Proposition 1, a $(35, 12)$-cluttered ordering exists for K_{71}. To generalize, suppose that we are to form an $(e, 2\ell)$-cluttered ordering of K_{2e+1} and that $e \leq \ell^2$. We could begin with $G(\ell, 1)$, and preserve any e of the edges. Then if the resulting graph has a wrapped ρ-labelling, the ordering is easily generated. The only potential difficulty here is in determining which e edges to retain. Instead, we can label the vertices of $G(\ell, 1)$, not removing any edges, so that every nonzero difference modulo $2e+1$ is represented as the edge difference on *at least* one edge; call this a *wrapped near-ρ-labelling* of $G(\ell, 1)$ modulo $2e+1$. If a wrapped near-ρ-labelling exists, we can omit edges one by one whose difference is multiply represented, until every difference is represented exactly once. Thus we are asking, in general, for wrapped near-ρ-labellings modulo v for $v \leq 2\ell^2 + 1$. Indeed, there is no requirement that the modulus be odd; when it is even, an edge difference of half the modulus generates *two* copies of each edge, and hence half the edges of this difference can be omitted and the other half retained. A wrapped near-ρ-labelling modulo $2e+2$ yields a cluttered ordering for e edges. Space does not permit a full exploration here, so we content ourselves with some examples. $G(7, 1)$ has no wrapped ρ-labelling (modulo 99), but we have found wrapped near-ρ-labellings for every odd modulus from 51 to 93 inclusive, and for every even modulus from 52 to 88 inclusive. This

establishes that for $25 \leq e \leq 46$, there is an $(e, 14)$-cluttered ordering of K_{2e+1}, and for $25 \leq e \leq 43$, there is an $(e, 14)$-cluttered ordering of K_{2e+2}.

The labellings all have a similar structure, with $\rho(v_{i+1,1}) - 1 = \rho(v_{i,0})$ for all i, and $\rho(v_{6,0}) = 0$. We therefore present only the labels on $\{x_{i0}\}$ for $0 \leq i \leq 5$ in the following:

v	$\rho(v_{i0})$	v	$\rho(v_{i0})$	v	$\rho(v_{i0})$	v	$\rho(v_{i0})$
52	2 4 6 10 37 25	53	2 4 6 8 12 38	54	2 4 6 11 26 39	55	2 4 6 8 15 26
56	2 4 6 11 40 27	57	2 4 6 8 16 27	58	2 4 6 12 28 42	59	2 4 6 8 17 28
60	2 4 6 12 43 29	61	2 4 6 8 18 29	62	2 4 6 13 30 45	63	2 4 6 8 19 30
64	2 4 6 13 46 31	65	2 4 6 8 19 41	66	2 4 6 14 32 48	67	2 4 6 8 21 32
68	2 4 6 14 49 33	69	2 4 6 11 33 56	70	2 4 6 15 34 51	71	2 4 6 11 57 34
72	2 4 6 15 52 35	73	2 4 6 12 59 35	74	2 4 7 17 54 44	75	2 4 6 15 26 44
76	2 4 13 20 43 51	77	2 4 6 17 26 46	78	2 4 13 44 53 60	79	2 4 7 26 36 67
80	2 4 14 21 45 54	81	2 4 8 27 69 38	82	2 4 46 55 15 22	83	2 4 8 38 70 28
84	2 22 5 15 57 47	85	2 4 8 28 39 72	86	8 28 71 3 51 41	87	2 4 33 71 11 47
88	10 13 18 62 34 57	89	2 6 18 36 27 50	91	2 9 30 68 14 49	93	2 9 14 31 69 50

This strategy extends naturally to $G(h, w)$ and to $\Gamma(h, w)$; when $w > 1$, we insist that when edge $\{v_{i,k}, v_{j,k+1}\}$ is removed, all edges of the form $\{v_{i,\ell}, v_{j,\ell+1}\}$ for $0 \leq \ell < w$ are removed, in order to ensure that the resulting ordering is $(e, 2h)$– or $(e, 2h+1)$–cluttered.

4 Large Windows

In this section, we examine the opposite extreme, when all but one vertex is induced within a window. An upper bound is easily proved:

Proposition 7. $c_{n-1}(n) \leq \lfloor \binom{n}{2}/2 \rfloor - 1$.

Proof. We must establish that when $d = \lfloor \binom{n}{2}/2 \rfloor$, every ordering π has d-maximum access cost equal to n. Suppose to the contrary that there is an ordering π with cost less than n. Let $G = K_n$, and consider $G_0^{\pi,d}$ and $G_{\lceil \binom{n}{2}/2 \rceil}^{\pi,d}$. These graphs must have a different isolated vertex, since at most one edge of K_n fails to appear in either; suppose that 0 and 1 are, respectively, the missing vertices. When $\binom{n}{2}$ is even, all edges appear in one of the two graphs and yet $\{0, 1\}$ appears in neither, which is a contradiction. So it remains to examine the case when $\binom{n}{2}$ is odd. Then $\{0, 1\}$ must be $e_{\pi(\lfloor \binom{n}{2}/2 \rfloor)}$. Now any graph $G_i^{\pi,d}$ for $1 \leq i \leq d$ contains the edge $\{0, 1\}$ and hence must miss a different vertex. Suppose that $G_1^{\pi,d}$ is missing vertex x, and $G_d^{\pi,d}$ is missing y. It cannot happen that $x = 0$, or all occurrences of 0 lie among the last d positions and hence all vertices appear there as well. By the same token, we find that $\{x, y\} \cap \{0, 1\} = \emptyset$. Since $G_0^{\pi,d}$ misses 0 and $G_d^{\pi,d}$ misses y, the last edge in the ordering must be $\{0, y\}$; in the same way, the first must be $\{1, x\}$. If $\{x, y\}$ is an edge, there is then no position in which to place it in the ordering. Therefore $x = y$. But then

since $G_1^{\pi,d}$ and $G_d^{\pi,d}$ both miss x, the total number of occurrences of x in edges is at most two, the final contradiction. □

Now we prove a matching upper bound.

Proposition 8. $c_{n-1}(n) \geq \lfloor \binom{n}{2}/2 \rfloor - 1$.

Proof. We define an ordering π on the edges of K_n. Let 0, 1, 2, and 3 be specific vertices of K_n, and let A be the set of remaining $n-4$ vertices. Edges are ordered as follows: {2,3}, {0,2}, {0,3}; $\{3, a\}$ for $a \in A$; $\{0, a\}$ for $a \in A$; X; {0,1}; Y; $\{1, a\}$ for $a \in A$; $\{2, a\}$ for $a \in A$; {1,2}, {1,3}. Here X and Y partition the remaining edges arbitrarily, subject to the condition that Y contains at least as many edges as does X, and contains at most one more edge. It is then easily verified that the graphs $G_i^{\pi,d}$ with $d = \lfloor \binom{n}{2}/2 \rfloor - 1$ miss first vertex 1, then 2, then 3, and finally 0. □

For $n = 8$, here is an example ordering: 23, 02, 03, 34, 35, 36, 37, 04, 05, 06, 07, 45, 46, 47, 01, 56, 57, 67, 17, 16, 15, 14, 27, 26, 25, 24, 12, 13. All sets of 13 consecutive edges only involve seven of the vertices. In fact, the only set of 14 consecutive edges involving all eight vertices is the last.

5 Inflation

While the techniques of the two preceding sections often give useful cluttered orderings for the RAID application, they can be difficult to apply to specific values of n and d. We therefore develop a simple technique, *inflation*, for providing reasonable orderings in cases where our earlier techniques do not apply directly. Suppose we wish to produce an edge ordering for K_n, and we have (d, f)-cluttered orderings for a collection of smaller complete graphs. If there is a (d, f)-cluttered ordering of K_t for $t < n$, write $n = st + r$, with $0 \leq r < t$. Form a complete graph K_n, and partition its vertices into t classes, each having size s or $s + 1$. Call the classes $X_1, \ldots, X_s$. For each $1 \leq i \leq s$, the edges within class X_i are partitioned into $t - 1$ classes, $I_{i,1}, \ldots, I_{i,t-1}$, so that the number of edges within the edge classes are as equal as possible. Then each has at least $\lfloor \binom{|X_i|}{2}/(t-1) \rfloor$ *edges*. Now form a (d, f)-cluttered ordering of K_t on vertex set $\{1, \ldots, t\}$. We order the edges of K_n as follows. For each edge $\{i, j\}$ of K_t in turn, we list all edges of K_n with one endpoint in X_i and the other in X_j (in any order). Then if this is the ℓth occurrence of i in an edge, we list the edges in $I_{i,\ell}$. We repeat this for j. The resulting ordering of K_n has 'inflated' every edge of K_t into a collection of at least $s^2 + 2\lfloor \binom{s}{2}/(t-1) \rfloor$ edges. Hence we have proved:

Proposition 9. *If a (d, f)-cluttered ordering of K_t exists, a (d', f')-cluttered ordering of K_{st+r} also exists, where $f' = \min(sf + f, sf + r)$ and $d' \geq (s^2 + 2\lfloor \binom{s}{2}/(t-1) \rfloor) \cdot d$.*

A very special case arises when $n = t + 1$. Then we find that $c_f(n) \geq c_{f-1}(n-1)$.

We do not expect inflation to yield edge orderings which actually minimize the number of vertices induced. Rather, it affords us a method for employing the specific solutions found by combinatorial methods (as in earlier sections) to produce acceptable orderings for a large range of relevant parameters.

6 Concluding Remarks

The impact of improved edge orderings on performance is treated in [8]. In this paper, we have focussed on combinatorial methods for producing cluttered orderings. We extended the notion of ladder orderings (which handle very small reads) to larger reads by establishing the existence of numerous wrapped labellings. The specific examples produced are of value in erasure codes of practical size. We also established an exact result when all but one of the vertices are induced within a window. Finally, we described a simple technique for employing existing orderings for small complete graphs to develop orderings for larger complete graphs; again, this can be expected to have practical value in finding reasonable (but almost certainly not optimal) orderings for particular parameters in a RAID system.

We have treated only the complete graph here, and it is natural to ask about orderings of other graphs as well. In the RAID application, we expect that simply removing the trailing edges from an ordering provides an erasure code with fewer information disks that enjoys similar performance benefits. However, considerations such as reliability and balanced ordering (see [10]) may argue against the omission of those particular edges; hence it may be of interest to extend the techniques here to handle a broader class of graphs.

Acknowledgments

Research of the authors is supported by the Army Research Office (U.S.A.) under grant number DAAG55-98-1-0272 (Colbourn). We thank Alan Ling and Alex Rosa for helpful comments.

References

1. Juraj Bosák. *Decompositions of Graphs*. Kluwer Academic Publishers, Dordrecht, 1990.
2. Yeow Meng Chee, Charles J. Colbourn and Alan C. H. Ling. Asymptotically optimal erasure-resilient codes for large disk arrays. *Discrete Applied Mathematics* 102 (2000), 3–36.
3. Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz and David A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys* 26 (1994) 145–185.
4. Myra B. Cohen. Performance analysis of triple erasure codes in large disk arrays. Master's thesis, University of Vermont, 1999.
5. M.B. Cohen and C.J. Colbourn. Optimal and pessimal orderings of Steiner triple systems in disk arrays. *Theoretical Computer Science*, to appear.

6. M.B. Cohen and C.J. Colbourn. Steiner triple systems as multiple erasure codes in large disk arrays. *Proceedings of IPCCC 2000 (19th IEEE International Conference on Performance, Computing and Communications)*, 2000, pp. 288-294.
7. M.B. Cohen and C.J. Colbourn. Ladder orderings of pairs and RAID performance. submitted for publication, 2000.
8. M.B. Cohen and C.J. Colbourn. Ordering edges for double erasure codes. *Proceedings of the Symposium on Parallel Algorithms and Architectures*, Crete, July 2001, to appear.
9. Garth A. Gibson. *Redundant Disk Arrays, Reliable Parallel Secondary Storage.* MIT Press, 1992.
10. Lisa Hellerstein, Garth A. Gibson, Richard M. Karp, Randy H. Katz and David A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica* 12 (1994) 182–208.
11. Mark C. Holland. *On-Line Data Reconstruction In Redundant Disk Arrays.* PhD thesis, Carnegie Mellon University, 1994.
12. Edward K. Lee. Software and performance issues in the implementation of a RAID prototype. Technical Report CSD-90-573, University of California at Berkeley, 1990.
13. Edward K. Lee. *Performance Modeling and Analysis of Disk Arrays.* PhD thesis, University of California at Berkeley, 1993.
14. Paul Massiglia. *The RAID Book, A Storage System Technology Handbook, 6th Edition.* The RAID Advisory Board, 1997.
15. Alexander Rosa. On certain valuations of the vertices of a graph. *Theory of Graphs (Internat. Sympos., Rome, 1966)*, Gordon and Breach, New York, 1967, pp. 349–355.
16. K. Salem and H. Garcia-Molina. Disk striping. In *Proceedings of the 2nd International Conference on Data Engineering*, pages 336–342, Washington, D. C., 1986. IEEE.
17. Daniel Stodolsky, Garth Gibson, and Mark Holland. Parity logging: Overcoming the small write problem in redundant disk arrays. *Computer Architecture News* 21 (1993), 64-75.

Improved On-Line Stream Merging: From a Restricted to a General Setting

Wun-Tat Chan, Tak-Wah Lam, Hing-Fung Ting*, and Wai-Ha Wong

Department of Computer Science, University of Hong Kong, Hong Kong
{wtchan,twlam,hfting,whwong}@cs.hku.hk

Abstract. Stream merging is a promising technique for reducing server bandwidth in video-on-demand systems. There are many heuristics for the problem proposed whose effectiveness has been confirmed empirically. However, it is desirable to prove their effectiveness mathematically. In the pioneering work [2], Bar-Noy and Ladner studied stream merging using competitive analysis. They designed an $O(\log n)$-competitive on-line scheduler, where n is the total number of stream requests. However, their result is applicable only to systems with large client bandwidth and buffer size. In this paper we design the first on-line scheduler for stream merging in the general setting, in which we lift the large resource requirements, and our scheduler achieves constant competitive ratio.

1 Introduction

We study an on-line scheduling problem arising from video-on-demand (VOD) systems that support stream merging. A VOD system often receives many requests for a hot video over a short period of time (say, Friday 7 pm to 9 pm). If the video server uses a dedicated video stream to serve each request, the bandwidth requirement for the server becomes enormous. A simple alternative is to batch [1,5] the requests so that a single stream is issued to serve requests arrived over a period of time; however, this increases the response time substantially.

To reduce the bandwidth requirement without sacrificing the response time, Eager *et al.* [6,7] proposed a promising technique called stream merging, which assumes that a client is equipped with extra buffer and can receive data from two streams simultaneously. The technique takes advantage of the assumptions as follows. Initially, a client receives data from a dedicated stream X for immediate playback. After some time, the client may start to buffer data simultaneously from another stream Y initiated earlier (due to another client). Suppose Y is initiated t time units before X. Then after listening to both X and Y for t time units, the client no longer needs X as it can play back the next t time units using the data in the buffer, while further buffering data from Y simultaneously. Hence, X may terminate now. We say that X *merges* with Y when the client starts to count on Y only. Merging streams allows several clients to be served using only one full stream plus a number of partial streams, thus saving the bandwidth of the server. Figure 1 shows an example.

* Research supported by the Hong Kong Research Grants Council HKU7103/99E.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 432–442, 2001.
© Springer-Verlag Berlin Heidelberg 2001

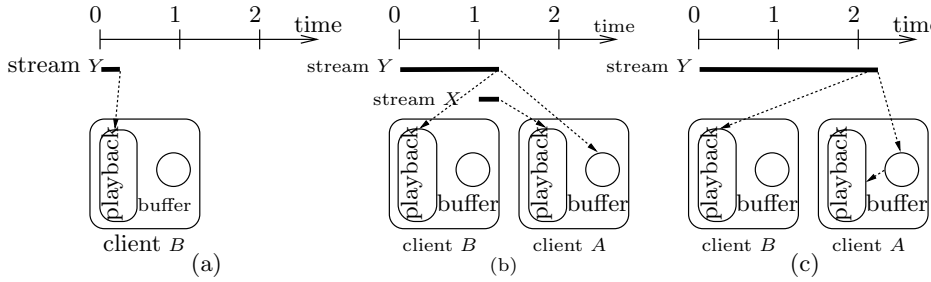


Fig. 1. (a) At time 0, client B is served by stream Y. (b) At time 1, client A arrives. It is served by stream X; it simultaneously buffers data from Y. (c) At time 2, A plays the content of its buffer and continues buffering from Y. X terminates. Assume the video length is 120 time units. The required bandwidth is 121 instead of 240.

An on-line scheduler for stream merging determines whether each existing stream should continue to run or merge with another earlier stream without the knowledge about the requests not yet arrived. The goal is to minimize the total bandwidth used, i.e., the total duration of all streams. Such schedulers were first studied in a primitive form in [3,9,10], in which a stream is allowed to merge with a full stream only. Eager *et al.* [6–8] gave a general on-line scheduler that performs well empirically. Bar-Noy and Ladner [2] was the first to study the problem using competitive analysis. An on-line scheduler is said to be *c-competitive*, or has *competitive ratio* at most c, if for any sequence I of requests, it uses at most c times of the bandwidth used by the optimal off-line schedule in serving I. In [2], Bar-Noy and Laner gave an on-line scheduler with competitive ratio at most $\min\{\log_\phi n, \log_\phi \frac{1}{2D}\} + O(1)$, where $\phi = (1+\sqrt{5})/2$, and $D < 1$ is the guaranteed startup delay measured as a fraction of the time for a full stream. In [4], we gave the first on-line scheduler with constant competitive ratio.

All the above-mentioned results assume that the total client bandwidth is at least twice the playback bandwidth (half of it is for normal playback and the additional half is for buffering an earlier stream) and the client buffer is sufficient to store up to half the size of a video. Eager *et al.* [8] pointed out that in reality, clients may have limited bandwidth and buffer. For the general problem–the $(\omega, \mathcal{B})$-stream merging problem, in which the client additional bandwidth is only a fraction ω ($0 < \omega \leq 1$) of the playback bandwidth and the client can buffer at most $\mathcal{B}$ ($\mathcal{B} > 0$) time units of video, only empirical results are presented (see [8]). In [2], Bar-Noy and Ladner leaved as an open problem whether the requests can be handled competitively in this general setting.

In this paper we give the first on-line scheduler for $(\omega, \mathcal{B})$-stream merging with constant competitive ratio (precisely, 5) for any given $0 < \omega \leq 1$ and any $\mathcal{B} > 0$, and thus solving the open problem of Bar-Noy and Ladner. Note that the competitiveness of our scheduler does not deteriorate even if the bandwidth and buffer are limited. Our result is based on a novel observation that the behavior

of any scheduler can be modeled by rectilinear (binary) trees on a triangular grid, and we will give the details in Section 2. In Section 3 we present a general methodology for constructing a competitive on-line scheduler from some near-optimal rectilinear tree. In Section 4 we show how to construct such near-optimal rectilinear tree.

2 Concepts and Notations

2.1 Preliminaries

In our discussion, time is divided into intervals of unit length, which can be arbitrarily small. Requests arriving within the same interval are treated as a single request and they are served by multicasting one stream. We suppose both video length $\mathcal{L}$ and buffer size $\mathcal{B}$ are measured in terms of time units.

Note that with the additional bandwidth being only a fraction ω of the playback bandwidth, it takes longer time for a stream X to merge with a stream Y initiated t time units earlier. It is not diffcult to see that X takes $\lceil 1/\omega \rceil t$ time units to buffer data from Y before the two streams can merge (see [8]). We call $\mathcal{C} = \lceil 1/\omega \rceil$ the *catch up ratio*. Observe that for X merging to Y successfully, the buffer size $\mathcal{B}$ could be equal to t, and since X must merge with Y before Y terminates, we require $t + \mathcal{C}t \leq \mathcal{L}$, or equivalently $t \leq \mathcal{L}/(1+\mathcal{C})$. Thus, $\mathcal{B}$ can be as small as $t \leq \mathcal{L}/(1+\mathcal{C})$. On the other hand, if $\mathcal{B}$ is smaller than $\mathcal{L}/(1+\mathcal{C})$, the optimal off-line schedule has only limited choices to merge the streams, and an on-line scheduler can simulate the off-line schedule easily (details given in the full paper). Thus, we only need to consider the case where $\mathcal{B} = \mathcal{L}/(1+\mathcal{C})$.

A request sequence is said to be *compact* if the last request arrives within $\mathcal{B}$ time units from the first request. Notice that such a sequence contains at most $\mathcal{B}+1$ requests. Intuitively, all streams initiated for a compact request sequence can merge successfully before the first stream terminates, and the first stream initiated must be a full stream. In fact, using an observation of Bar-Noy and Ladner [2], we can show that for compact request sequences, it suffices to focus on schedules with the following property:

> The first stream is the only full stream and all other streams merge with other streams eventually.

Since the first stream always contributes a dominating portion in the overall bandwidth usage of any schedule, the overall bandwidth of different schedules may not differ a lot. To make a more meaningful comparison between different schedulers, we define the *full cost* to be the bandwidth used by all the streams, and define the *merge cost* to be the bandwidth used by all except the first stream. Note that the full cost and merge cost differ by exactly the video length $\mathcal{L}$.

Compact request sequence is a key notion for stream merging because a general request sequence can be processed competitively if each consecutive compact request sequence can be processed competitively. More precisely, suppose $\mathcal{A}$ is an on-line scheduler for the problem such that with respect to compact request

sequences, its merge cost is at most c times that of the optimal off-line schedule. We can use $\mathcal{A}$ to process arbitrary request sequence as follows. As requests arrive, we divide them into groups. When a request arrives within $\mathcal{B}$ time units from the first request of the current group, it is inserted to this group, and we use $\mathcal{A}$ to schedule a stream for it and determine how the stream merges with other streams. Otherwise, we start a new group and initiate a full stream for this request. Notice that each group of requests is in fact a compact request sequence. The following theorem, which generalizes the one in [2], asserts that the bandwidth used by the resulting schedule is at most $c + \mathcal{L}/(\mathcal{L} - \mathcal{B})$ times that of the optimal off-line schedule. The proof will be given in the full paper.

Theorem 1. *Let $\mathcal{A}$ be an on-line scheduler such that with respect to compact request sequence, its merge cost is at most c times that of the optimal off-line schedule. Then for any arbitrarily long request sequence, using $\mathcal{A}$ to schedule separately each compact request sequence can produce a schedule with an overall bandwidth at most $c + \mathcal{L}/(\mathcal{L} - \mathcal{B})$ times that of the optimal off-line schedule.*

In the rest of this paper, we focus on compact request sequences and give an on-line scheduler whose merge cost is at most three times that of the optimal off-line schedule. Then, by Theorem 1, the competitive ratio of this scheduler is at most $3 + \mathcal{L}/(\mathcal{L} - \mathcal{B}) = 5$ because $\mathcal{C} \geq 1$, and we are consider the case $\mathcal{B} = \mathcal{L}/(1 + \mathcal{C})$. Unless specified otherwise, all request sequences mentioned are meant to be compact request sequences. To ease discussion, we assume that the first request of all compact request sequences arrives at time 0.

2.2 Binary Merge Trees

In this section we show how to model schedules using binary merge trees, which attempt to model the occurrences of merging and the status of each stream.[1] Below, we use the notation $r(s)$ to denote the time when a stream s is initiated.

A binary merge tree is a rooted, ordered binary tree. For any node with (out-)degree one, we require the node to have a left son but no right son. Every node u is associated with a stream s_u and a time t_u. In particular, every leaf ℓ corresponds to a distinct stream that is initiated due to a request, and $t_\ell = r(s_\ell)$.

For an internal node u, its associated time is always greater than that of its sons, and its stream is always inherited from its left son. If u is of degree two, u and its right son, denoted by v, represent the occurrence of a merging. We require that $r(s_v) > r(s_u)$ and $t_u = t_v + \mathcal{C}(r(s_v) - r(s_u))$. At time t_v, the stream s_v (precisely, the clients served by s_v) starts to buffer the content of s_u; after $\mathcal{C}(r(s_v) - r(s_u))$ time units (i.e., at time t_u), s_v merges with s_u and terminates.

Regardless of whether u is degree one or two, u and its left son w corresponds to a continuation of the same stream from time t_w to time t_u. That is, the stream has no plan to merge with another earlier stream during this interval. Note that the root as well as all the nodes on the leftmost path of the tree are associated

[1] This is in contrast to the merge trees used by Bar-Noy and Ladner [2], which model the relationship among the streams.

with the same stream, which does not merge with any stream and runs till the end. The time associated with the root is exactly $\mathcal{L}$.

The *cost* of an edge (u, v) is defined as $t_u - t_v$. The *full cost* of a binary merge tree is the sum of the cost over all edges, and the *merge cost* is the full cost minus the cost of the edges on the leftmost path.

Recall that we focus on compact request sequences and schedules in which all streams merge with the first one. Note that each binary merge tree corresponds to one of these schedules with the same full cost and merge cost, and vice versa. Thus, for any compact request sequence, there is a binary merge tree with full cost and merge cost equal those of the optimal off-line schedule. Below, we show that the optimal binary merge tree, the tree associated with an optimal off-line schedule, can be assumed to have a more uniform structure.

Lemma 1. *Consider any compact request sequence I. There is an optimal binary merge tree for I such that all the internal nodes, except possibly the root, have both left and right sons.*

Proof. Let T be the optimal binary merge tree with the minimum number of nodes. Assume there is a degree one internal node v. By definition, v has a left son, say w. Let u be the parent of v. If v is the left son of u, we remove v and add an edge between u and w. The cost is not changed; contradicting that T has the minimum number of nodes. If v is the right son of u, s_v starts buffering from s_u at t_v. We make s_v starts buffering from s_u earlier, say at t_w. This decreases the merge cost of the schedule and contradicts the optimality of T. □

2.3 Rectilinear Trees

This section discusses a powerful tool called *rectilinear trees* to model schedules of stream merging. The new tool is more useful for visualizing the difference between an arbitrary schedule and the optimal off-line schedule. Roughly speaking, rectilinear trees are binary merge trees embedded on a triangular grid. Let $K = \mathcal{B} + 1$. Consider a $K \times K$ grid with K *grid points* (i.e., with $K - 1$ *grid arcs*) on each boundary. The grid is distorted into a rectangle: The horizontal grid arc has a length of $(1 + \mathcal{C})$ and the vertical grid arc a length of $\mathcal{C}$.

A *grid line* is a sequence of consecutive horizontal (or vertical) grid arcs. A *grid path* is a sequence of grid arcs connecting any two grid points. The rows and columns of the grid are numbered 0 to $K - 1$ from top to bottom, and from left to right, respectively. The *triangular grid* in concern is the upper right triangle with grid points $(0, 0), (1, 1), \cdots, (K - 1, K - 1)$ as the hypotenuse.

A *rectilinear tree* is a binary tree on the triangular grid satisfying the following properties. (1) The root is at the top rightmost corner of the triangular grid and the leaves are on the hypotenuse of the triangular grid. (2) Any node and its parent are either on the same row or on the same column, i.e., any edge is either a horizontal or a vertical grid line. (3) The grid path from any leaf to the root is *monotone*—The rectilinear distance to the root is strictly decreasing as we traverse from the leaf to the root.

For any rectilinear tree T, we define the *length* of an edge to be the length of the corresponding horizontal or vertical grid line. The *full length* of T is the total length of all its edges, and the *branch length* is the total length of all edges except those edges along the leftmost path of the tree. Notice that two rectilinear trees with the same set of grid arcs have the same branch length even if the set of nodes may differ. Branch length is a notion heavily used below; to simplify our discussion, we denote by $\delta(T)$ the branch length of T. The following theorem shows the correspondence between binary merge trees and rectilinear trees.

Theorem 2. *Any binary merge tree can be embedded on a triangular grid as a rectilinear tree with branch length equals the merge cost of the binary merge tree.*

Proof. Let T be a binary merge tree. For any node u in T, we create a node u_R at the grid point $(r(s_u), (t_u + \mathcal{C}r(s_u))/(1+\mathcal{C}))$. If u is a leaf, u_R is on the hypotenuse. The root becomes a node at the upper rightmost corner of the grid. If u is an internal node, let w be the left son of u. w_R is at $(r(s_w), (t_w + \mathcal{C}r(s_w))/(1+\mathcal{C}))$. As s_u is the same as s_w and $t_u > t_w$, u_R lies on the right of w_R on the same row. Thus, the edge (u, w) becomes a horizontal grid line between u_R and w_R, with length $(1+\mathcal{C}) \times (t_u + \mathcal{C}r(s_u) - t_w - \mathcal{C}r(s_w))/(1+\mathcal{C}) = t_u - t_w$, which equals the cost of the edge (u, w) in T. The case for the right son can be proved similarly. The branch length of the rectilinear tree equals the merge cost of T. □

3 The On-line Scheduler

A crucial issue in designing on-line scheduler is to estimate the cost of the optimal solution. We give a simple way to estimate the cost of the optimal solutions of *all* compact request sequences using only one rectilinear tree, called the *complete rectilinear tree*. This rectilinear tree when restricted to any compact request sequence I gives an attainable estimate of the branch length of the optimal rectilinear tree for I. Our on-line scheduler makes reference to this tree to schedule the merging of streams; the resulting schedule corresponds to a rectilinear tree with branch length always upper bounded by the estimated branch length of the optimal rectilinear tree.

Define a *complete rectilinear tree* T to be a rectilinear tree whose leaves comprise all the grid points along the hypotenuse of the triangular grid. For any compact request sequence I, for simplicity, I also denotes the subset of grid points along the hypotenuse corresponding to the requests. The rectilinear tree T when restricted to I is called the *partial rectilinear tree* of T with respect to I. It is a rectilinear tree denoted by $T\|_I$ whose set of leaves is I and set of grid arcs is a subset of that of T. In other words, for any grid arc e in T, if e is along a path of T from the root to some leaves in I, e is in $T\|_I$; otherwise, e is not in $T\|_I$. Note that $T\|_I$ is unique for any given I.

In this section, we describe a methodology which makes reference of a given complete rectilinear tree T to derive an on-line algorithm $\mathcal{A}_T$. For any compact request sequence I, $\mathcal{A}_T$ gives a schedule whose corresponding rectilinear tree, denoted by $\mathcal{A}_T(I)$, has a branch length at most that of $T\|_I$. That means for

any given T and I we can produce a schedule with merge cost at most $\delta(T\|_I)$. On the other hand, there is an optimal binary merge tree for I satisfying the property stated in Lemma 1. By Theorem 2, this tree can be embedded on the triangular grid as a rectilinear tree, denoted by OPT_I. In Section 4 we describe a particular complete rectilinear tree S such that $\delta(S\|_I) \leq 3\,\delta(OPT_I)$, and hence the merge cost of the schedule produced by $\mathcal{A}_S$ is at most three times that of the optimal off-line schedule. Together with Theorem 1, we have the next theorem.

Theorem 3. *$\mathcal{A}_S$ is 5-competitive.*

In this section we assume T is a chosen complete rectilinear tree. We give an on-line scheduler $\mathcal{A}_T$ which constructs a rectilinear tree based on T and a compact request sequence. The scheduler makes reference to T to determine how the streams serving the requests merge with each other. Details of $\mathcal{A}_T$ are as follows.

When the first request arrives (at time 0), the scheduler initiates a stream for this request and this stream will not merge with any stream. The initial rectilinear tree contains only two nodes, namely, the root at the top rightmost corner and a leaf at the top leftmost corner, of the triangular grid. These two nodes are connected by a horizontal grid line. All the grid points along this grid line are associated with the first stream.

When a subsequent request arrives, it is assigned a dedicated stream X immediately. We also determine when X should begin buffering data from which earlier stream. Let (r, r) denote the leaf corresponding to current request where r is its arrival time. We locate in T the least common ancestor a of (r, r) and the leaf corresponding to the request just before the current request. A new grid path is added to the current rectilinear tree as follows.

If a lies on column r, the new grid path is a vertical grid line running upwards from (r, r). The path stops at the first grid point it intersects with the current rectilinear tree. In this case, X immediately buffers data from the stream associated with the intersecting grid point.

If a does not lie on column r, it must lie on the right of column r because of the monotone property of the paths in T. Suppose a lies on column c_a. The new grid path comprises a horizontal grid line from (r, r) to (r, c_a) and a vertical grid line from (r, c_a) to the first grid point it intersects with the current rectilinear tree on column c_a. In this case, $(1 + \mathcal{C})(c_a - r)$ time units after X is initiated, X starts buffering from the stream associated with the intersecting grid point.

All the grid points along the new path, except the intersecting point, are associated with X. The decision is made every time a request arrives. For any compact request sequence I, the next lemma bounds $\delta(\mathcal{A}_T(I))$ by $\delta(T\|_I)$.

Lemma 2. *For any complete rectilinear tree T and compact request sequence I, $\delta(\mathcal{A}_T(I)) \leq \delta(T\|_I)$.*

Proof. (Sketch.) Suppose I has k requests. For $1 \leq i \leq k$, let I_i be the first i requests in I, and $\mathcal{A}_T^i$ be the rectilinear tree constructed by $\mathcal{A}_T$ for I_i. We claim that $\mathcal{A}_T^i$ is always on the right of $T\|_{I_i}$, precisely, for any grid point $(r, c) \in T\|_{I_i}$,

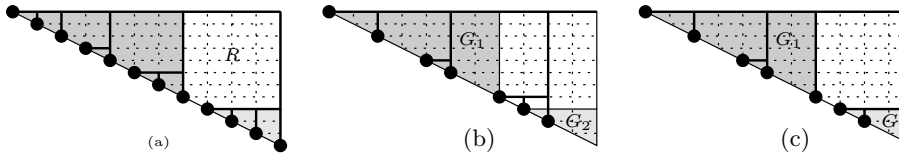


Fig. 2. Suppose $\mathcal{C} = 1$. A 12×12 triangular grid G is divided into an 8×8 and a 4×4 triangular grids G_1 and G_2. (a) The complete rectilinear tree S on G. The unshaded area is the rectangle R with top rightmost and the bottom leftmost corners at $(0, 11)$ and $(8, 7)$. (b) An optimal rectilinear tree for a given set of compact request sequence with seven requests. (c) After a recursive step the grid arcs within the rectangle (unshaded area) are removed and the grid arcs on the four boundaries are included.

there exists a grid point $(r', c) \in \mathcal{A}_T^i$ with $r' \geq r$. Since the path in T is monotone, the grid path added to $\mathcal{A}_T^{i-1}$ by $\mathcal{A}_T$ for the i-th request has a length at most that of the path added to $T\|_{I_{i-1}}$ to form $T\|_{I_i}$. Thus, $\delta(\mathcal{A}_T(I)) \leq \delta(T\|_I)$ as they are the sums of the corresponding path lengths. □

4 A Good Complete Rectilinear Tree

In this section we describe a complete rectilinear tree S on a $K \times K$ triangular grid Δ, which provides a good estimate of all optimal rectilinear trees. Precisely, we show that $\delta(S\|_I) \leq 3\,\delta(OPT_I)$ for all compact request sequences I.

We construct S recursively. For any integer n, let $f(n) = \lfloor((1 + \mathcal{C})/(1 + 2\mathcal{C}))(n-1)\rfloor + 1$. The root of S is at $(0, K-1)$ of Δ. All the K grid points along the hypotenuse are the leaves of S. If S has only one leaf (i.e., when $K = 1$), we are done. If S has at least two leaves, it has two subtrees. The left subtree has its root at $(0, f(K) - 1)$ and it contains the top $f(K)$ leaves along the hypotenuse. The right subtree has its root at $(f(K), K-1)$ and it contains the rest $K - f(K)$ leaves. These two subtrees are then constructed recursively, and their roots are connected to the root of S by a horizontal grid line with $K - f(K)$ grid arcs and a vertical grid line with $f(K)$ grid arcs. See Figure 2(a).

To prove $\delta(S\|_I) \leq 3\delta(OPT_I)$, we derive a procedure TRANSFORM that transforms OPT_I into a rectilinear tree O_I that uses only grid arcs of S, and whose set of leaves containing I. TRANSFORM is conservative in replacing the edges of OPT_I so that $\delta(O_I) \leq 3\delta(OPT_I)$. On the other hand, $S\|_I$ contains the minimal subset of grid arcs in S to form a rectilinear tree with I as leaves and using grid arcs of S only. Thus, $\delta(S\|_I) \leq \delta(O_I)$, and $\delta(S\|_I) \leq \delta(O_I) \leq 3\delta(OPT_I)$.

4.1 The Transformation

TRANSFORM is recursive; it takes two parameters G and T. G is a triangular grid, and T is a rectilinear tree embedded on G whose root is at the top rightmost corner of G. Initially, $G = \Delta$ and $T = OPT_I$. In the first step of TRANSFORM, consider the rectangle R whose top rightmost corner is at $(0, K-1)$ and bottom leftmost corner is at $(f(K), f(K) - 1)$.[2] See Figure 2(a). Note that S does not

[2] R, excluding its bottom leftmost corner, is enclosed entirely in Δ.

cross R; more precisely, no path in S enters R from a boundary of R, goes through at least one of its interior grid points, and leaves from a distinct boundary. Thus, we must remove all grid arcs of OPT_I within R.

If OPT_I has a path crossing R from the bottom boundary, we expand OPT_I to include all the arcs on the bottom and right boundaries of R, except the leftmost one (which is outside Δ). If OPT_I has a path crossing R from the left boundary, we include arcs on the left and top boundaries of R, except the bottommost one. We then remove all grid arcs that are within R. See Figure 2(b) and (c). New nodes are created on any new intersections of grid arcs introduced.

Let T_0 be the new tree formed after the above step. Ignoring the interior grid points of R, we partition Δ into two disjoint triangular grids: an $f(K) \times f(K)$ one G_1 to the left of R and a $(K - f(K)) \times (K - f(K))$ one G_2 below R. See Figure 2(c). Consider the part of T_0 in G_1. All paths in T_0 starting from the leaves in G_1 to the root of T_0 exit G_1 through its top rightmost corner. If this corner is not a node of T_0, we add a node there. Then, the part of T_0 in G_1 forms a tree T_1 that is embedded on G_1 with the root at its top rightmost corner. If G_1 is at least 3×3 and T_1 contains at least a leaf, we recursively call TRANSFORM on G_1 and T_1 to form a tree T_1' that uses only grid arcs of S and retains the corresponding leaves. We handle the part of T_0 in G_2 similarly, and let T_2' be the resulting tree. Note that T_1' and T_2', together with the tree edges of T_0 on the boundaries of R, form O_I.

4.2 Analysis of Procedure Transform

In this section we prove that $\delta(O_I) \leq 3\delta(OPT_I)$. Let E_a and E_r be the set of grid arcs added and removed by calling TRANSFORM on $G = \Delta$ and $T = OPT_I$. For any set of grid arcs E, we denote by $|E|$ the total length of grid arcs in E. We first show that $|E_a| \leq 3|E_r|$. Note that it suffices to show that in each recursive step (excluding the recursive calls), the total length of grid arcs added is at most three times that removed. Since the analysis is the same for every recursive step, we only consider the first step, in which the grid arcs in OPT_I within R are removed, where R is the rectangle whose top rightmost and bottom leftmost corners are at $(0, K-1)$ and $(f(K), f(K)-1)$ respectively. Let $\mathcal{E}_a$ and $\mathcal{E}_r$ denote the set of grid arcs removed and added in this particular step respectively.

Lemma 3. *If* OPT_I *crosses* R, OPT_I *contains a path crossing* R *vertically or horizontally, i.e., the path passes through a single column or a single row of* R.

Proof. Suppose OPT_I has a path σ from a leaf to the root that crosses R. σ exits R through either the right or top boundary. If σ exits through the right boundary, say on row i, σ or another path in OPT_I must go through R horizontally on row i. Otherwise, there is a node in OPT_I having at most one son. This contradicts Lemma 1 and Theorem 2. The other case can be proved similarly. □

If OPT_I does not cross R, obviously $|\mathcal{E}_a| \leq 3\,|\mathcal{E}_r|$; otherwise, we have two cases depending on whether OPT_I has a path crossing R vertically or horizontally.

Let $r(K)$ and $t(K)$ denote the length of the right and top boundaries of R respectively. In other words, $r(K) = \mathcal{C}f(K)$ and $t(K) = (1 + \mathcal{C})(K - f(K))$.

Lemma 4. *If* OPT_I *contains a path crossing* R *vertically,* $|\mathcal{E}_r| \geq r(K)$ *and* $|\mathcal{E}_a|-|\mathcal{E}_r| \leq r(K)+t(K)-1-2\mathcal{C}$. *If* OPT_I *contains a path crossing* R *horizontally,* $|\mathcal{E}_r| \geq t(K)$ *and* $|\mathcal{E}_a| - |\mathcal{E}_r| \leq r(K) + t(K) - 1 - 2\mathcal{C}$.

Proof. We only prove the first scenario, and the second one can be proved symmetrically. Let $H_{\mathtt{top}}$ and $H_{\mathtt{in}}$ be the sets of horizontal grid arcs in OPT_I that are on the top boundary of R and inside R, respectively. Thus, $|\mathcal{E}_r| \geq r(K) + |H_{\mathtt{in}}|$.

Suppose OPT_I has a path crossing R from the left boundary at grid point p. TRANSFORM expands OPT_I to include all four boundaries, excluding the two arcs adjacent to the bottom leftmost corner. Thus, $|\mathcal{E}_a| \leq 2r(K)+t(K)+(t(K)-|H_{\mathtt{top}}|) - 1 - 2\mathcal{C}$. Consider the path σ from p to the root of OPT_I. The total length of horizontal grid arcs in σ is at least $t(K)$, and is at most $|H_{\mathtt{top}}| + |H_{\mathtt{in}}|$. This implies $t(K) \leq |H_{\mathtt{top}}| + |H_{\mathtt{in}}|$. Therefore, $|\mathcal{E}_a| - |\mathcal{E}_r| \leq 2r(K) + t(K) + (t(K) - |H_{\mathtt{top}}|) - 1 - 2\mathcal{C} - (r(K) + |H_{\mathtt{in}}|) \leq r(K) + t(K) - 1 - 2\mathcal{C}$. The case for OPT_I not crossing R from the left boundary can be proved easily. □

By simple arithmetic, we have $r(K) \geq t(K) - 1 - \mathcal{C}$ and $t(K) \geq r(K) - \mathcal{C}$. By Lemma 4, $|\mathcal{E}_a| \leq 3\,|\mathcal{E}_r|$. Thus, $|E_a| \leq 3\,|E_r|$. Since E_a and E_r are disjoint, $|O_I| = |OPT_I| + |E_a| - |E_r|$. Note that $\delta(OPT_I) = |OPT_I| - \ell$ and $\delta(O_I) = |O_I| - \ell$, where ℓ is the length of the top boundary of Δ. Therefore, $\delta(O_I) = \delta(OPT_I) + |E_a| - |E_r|$. Since $|E_r| \leq \delta(OPT_I)$, $\delta(O_I) \leq \delta(OPT_I) + 2|E_r| \leq 3\delta(OPT_I)$.

Lemma 5. $\delta(O_I) \leq 3\delta(\mathrm{OPT}_I)$.

Recall that $\delta(S\|_I) \leq \delta(O_I)$. By Lemmas 2 and 5, we have $\delta(\mathcal{A}_S(I)) \leq \delta(S\|_I) \leq \delta(O_I) \leq 3\,\delta(OPT_I)$ and we have the following theorem.

Theorem 4. *Given any compact request sequence* I, $\delta(\mathcal{A}_S(I)) \leq 3\,\delta(\mathrm{OPT}_I)$.

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 253–258, 1996.

[2] A. Bar-Noy and R. E. Ladner. Competitive on-line stream merging algorithms for media-on-demand. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 364–373, 2001.

[3] Y. Cai, K. A. Hua, and K. Vu. Optimizing patching performance. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 204–215, 1999.

[4] W.-T. Chan, T.-W. Lam, H.-F. Ting, and W.-H. Wong. A 5-competitive on-line scheduler for merging video streams. In *Proceedings of the Fifteenth International Parallel and Distributed Processing Symposium*, pages 201 (full version of the paper in the accompanied CD–ROM), 2001.

[5] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 15–23, 1994.

[6] D. Eager, M. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. In *Proceedings of the Fifth International Workshop on Advances in Multimedia Information Systems*, pages 80–87, 1999.

[7] D. Eager, M. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In *Proceedings of the Seventh ACM International Conference on Multimedia*, pages 199–202, 1999.

[8] D. Eager, M. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-effective video-on-demand. In *Proceedings of Conference on Multimedia Computing and Networking*, pages 206–215, 2000.

[9] L. Gao and D. Towsley. Supplying instantaneous video-on-demand services using controlled multicast. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 117–121, 1999.

[10] K. A. Hua, Y. Cai, and K. Vu. Patching: A multicast technique for true video-on-demand services. In *Proceedings of the Sixth ACM International Conference on Multimedia*, pages 191–200, 1998.

On-Line Deadline Scheduling on Multiple Resources

Jae-Hoon Kim and Kyung-Yong Chwa

Department of Electrical Engineering & Computer Science,
Korea Advanced Institute of Science and Technology, Taejon 305-701, Korea,
{jhoon,kychwa}@jupiter.kaist.ac.kr

Abstract. We consider an on-line non-preemptive deadline scheduling on multiple resources. Each job has the slack time during which it can wait until its expiration time, the last time it can be started to be completed by the deadline. There are a number of resources on which the jobs are allocated. The effect of slack times on the on-line deadline scheduling was investigated in [8] by analyzing the performance of the Earliest EXpiration Time First (EXF) algorithm. But most of the previous work is restricted to a single resource, and in this paper, we propose a different method of analysis which can also be applied to multiple resources. So we can extend the result of [8] to multiple resources.

1 Introduction

Consider the following problem: we are given m resources and a set of job requests, where each job can be served by at most one of the resources and has the processing time for which it uses the resource and the deadline by which it should be completed, and each resource serves only one job at any time. Not all jobs are required to be scheduled and some jobs can be rejected but they cannot contribute to the throughput. Whenever a resource is idle, scheduling algorithms should determine which job to be processed on the resource so that it maximizes the total of gains of accepted jobs.

The schedule is *non-preemptive*, that is, once a job is scheduled, it cannot be preempted or rejected and it is *on-line*, where the algorithm has no knowledge of jobs until they arrive and when they arrive, the algorithm is informed the processing times and deadlines of the jobs. The performance of an on-line algorithm is estimated through *competitive analysis* [9, 12], in which it is compared with that of an off-line optimal algorithm that knows all informations of the future requests.

In this paper, the problem is described in terms of machine scheduling. Each job J_i has the release time r_i, the processing time p_i, and the deadline d_i and there are m identical machines on which jobs are scheduled. The *expiration time* x_i of a job J_i is defined to be the last time at which J_i can be scheduled to completion by the deadline, that is, $d_i = x_i + p_i$ and the *slack time* s_i of J_i to be $x_i - r_i$. If the slack time of a job is more than zero, before scheduling it, an on-line scheduling algorithm has a possibility of waiting for better jobs

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 443–452, 2001.
© Springer-Verlag Berlin Heidelberg 2001

which will arrive. Assume that for each job J_i, its slack time satisfies $s_i \geq \kappa p_i$ for some constant κ. Depending on the value of κ, we will investigate the effect of the slack time on the performance of the on-line scheduling algorithm. In [8], Goldwasser proposed the Earliest EXpiration Time First (EXF) algorithm that schedules the job with the smallest expiration time among all available jobs at all times. His work is restricted to a single machine, as in the previous results [7] and [10]. In [7] and [8], the authors divide the time *busy periods* during which there is no idle time on the machine in the schedule of EXF. Since jobs scheduled within a busy period have no effect on the schedules of the other busy periods, it is sufficient to analyze the performance of EXF only in a busy period. But this cannot be applied to multiple machines, where busy periods on distinct machines are aligned to cross at some times. We provide a different analysis that can also show the performance of EXF on multiple machines, where the time is divided into periods in a different way according to the behavior of EXF.

Let $\mathcal{S}$ be a set of job instances. Then $EXF(\mathcal{S})$ ($OPT(\mathcal{S})$) represents the sum of the processing times of jobs scheduled by EXF (OPT, resp.) in $\mathcal{S}$. Interchangeably, for a time interval T, $EXF(T)$ ($OPT(T)$) also represents the set of all jobs scheduled by EXF (OPT, resp.) in T. Also $|\mathcal{S}|$ is the number of jobs in $\mathcal{S}$.

1.1 Previous Work

Off-line non-preemptive scheduling of jobs with deadlines is well-known in the literature from the early days [6, 11], but very recently, there have been active studies for this problem [1, 2]. where the jobs have arbitrary release times and weights and the types of machines are given as the single machine, identical machines, and unrelated machines. They give constant approximation algorithms using various optimization techniques, for example, LP-rounding, local search, and etc.

In the on-line setting, it originates from the work of Lipton and Tomkins [10], where each job must be scheduled or rejected as soon as it arrives. For the general case in which the jobs have arbitrary processing times, they give $O((\log \Delta)^{1+\epsilon})$-competitive algorithm and show that no $O(\log \Delta)$-competitive algorithm can exist, where Δ is the ratio of largest to smallest processing time. In [7], they allowed the jobs to specify arbitrary slacks, i.e., $\kappa \geq 0$, and the effect of the positive slack times was shown in [8]. For equal processing time jobs, he proves EXF is $(1 + \frac{1}{\lfloor \kappa \rfloor + 1})$-competitive and shows that no on-line algorithm can have a better ratio, and for arbitrary processing time jobs, he gives a $(2 + \frac{1}{\kappa})$-competitive algorithm and shows that this is best possible for all deterministic on-line algorithms. But these results are all restricted to the model of a single machine.

For on-line preemptive scheduling problem, there are several related studies [3, 4, 5]. In [4], the effect of slack times for preemptive scheduling on a single machine is investigated, and in [5], the performance of preemptive scheduling on multiple machines is characterized by the stretch factor which is the ratio of the response time to the processing time, that is, $\frac{s_i + p_i}{p_i}$.

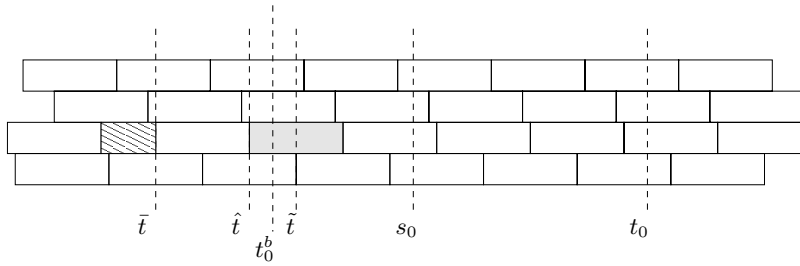


Fig. 1. The division of time intervals.

2 Equal Processing Time Jobs

2.1 Case $\lfloor \kappa \rfloor \geq 1$

Let $\mathcal{J}$ be the set of all job instances. W.l.o.g., we assume each job has the processing time 1. First we consider the case $\lfloor \kappa \rfloor \geq 1$. According to the behavior of EXF, we divide the time into periods and classify the jobs in $\mathcal{J}$. Here, observe the behavior of EXF. When a new job arrives, EXF puts it into a queue, and whenever there is an idle time on some machine, EXF selects the job with the earliest expiration time in the queue and schedules it on the idle machine. If a job has not been selected until its expiration time, it is rejected at that time.

Inductively, we define the time periods $T_i = [t_i^b, t_{i-1}^b)$, $i = 0, 1, \cdots$, and $t_{-1}^b = \infty$. Assume that the time periods $T_j, j = 0, \cdots, i$ have been defined and there is a job which EXF rejects in $(-\infty, t_i^b)$. Then we consider the job which is rejected in the last in $(-\infty, t_i^b)$ and denote it by I_i. Let t_i be the time at which I_i is rejected and let s_i be the release time of I_i. Then any job with expiration time $> t_i$ cannot be scheduled in $[s_i, t_i]$, because if such a job were scheduled, it would have an expiration time greater than I_i and EXF would schedule I_i. Thus if a job has an expiration time $> t_i$, either it is scheduled after t_i or before s_i. We refer to the time t_i as *the critical time*, and we will also define the time t_{i+1}^b, called *the barrier time*. Let $\bar{t}$ be the time at which an idle time last appears on some machine before s_i. If there is no job with expiration time $> t_i$ which is scheduled in $[\bar{t}, s_i)$, we set the time t_{i+1}^b to be $\bar{t}$. Otherwise, let $\hat{I}$ be the job with expiration time $> t_i$ which is last scheduled in $[\bar{t}, s_i)$ and let $\hat{t}$ be the time when $\hat{I}$ is scheduled. Then at the time $\hat{t}$, there is no job with expiration time $\leq t_i$ in the queue, because if such a job were in the queue, EXF would schedule it in place of $\hat{I}$. Consider the job, denoted by $\tilde{I}$, which is first scheduled after $\hat{t}$, and let $\tilde{t}$ be the time at which $\tilde{I}$ is scheduled. Then we can see that new jobs have been released in $(\hat{t}, \tilde{t}]$, because otherwise, if $\tilde{t} \geq s_i$, then it is a contradiction to the fact that I_i was released at s_i, and if $\tilde{t} < s_i$, then $\tilde{I}$ were also in the queue at $\hat{t}$, that is, $\tilde{I}$ had the expiration time $> t_i$, and it is a contradiction to the definition of $\hat{I}$. Thus we set t_{i+1}^b to be the time at which a job is first released in $(\hat{t}, \tilde{t}]$, and we call the half-open interval $[t_{i+1}^b, t_i^b)$ *the principal period* and denote it by T_{i+1}. See Figure 1. Then a principal period T_k is given for some $k \geq 0$, such

that either there is no job rejected in $[0, t^b_{k-1})$ or there is the job I_k last rejected in $[0, t^b_{k-1})$ but there is neither an idle time on some machine nor a job with expiration time $> t_k$ in $[0, s_k)$, where t_k and s_k are the critical time determined by I_k and the release time of I_k, respectively.

In the following, we will investigate the characteristics of the principal periods T_i, $i = 0, \cdots, k$.

Lemma 1. *In each principal period T_i, there is no idle time on any machine in EXF's schedule between the barrier time t^b_i and the critical time t_i, and each job scheduled in $[t^b_i, t_i]$ should have an expiration time $\leq t_i$.*

Proof. Fix a principal period T_i. Let I_i be the job which is rejected by EXF at t_i and s_i be the release time of I_i. From the definition of the barrier time t^b_i, we see that no job with expiration time $> t_i$ is scheduled in $[t^b_i, s_i)$ and no idle time exists on any machine in that interval. After I_i is released at s_i, no idle time appears on any machine, and EXF never schedules any job with expiration time $> t_i$ until I_i is rejected at t_i, because otherwise, EXF could have scheduled I_i.

In each principal period T_i, we call jobs in $EXF((t_i, t^b_{i-1}))$ $(OPT((t_i, t^b_{i-1})))$ *late* jobs of EXF (OPT, resp.) and $\mathcal{L}_{EXF}(T_i)$ $(\mathcal{L}_{OPT}(T_i))$ represents the set of all late jobs of EXF (OPT, resp.) in T_i. In the next lemma, it is shown that each late job of OPT should be scheduled by EXF and it is also a late job of EXF in some principal period.

Lemma 2. *In each principal period T_i, all late jobs of OPT in T_i must be scheduled by EXF and in particular, they are late jobs of EXF in some T_j.*

Proof. Fix a principal period T_i and let J be a late job of OPT in T_i. Then it has the expiration time $e > t_i$. Assume it was not scheduled by EXF before or at t_i. Then either, at t_i, J was in the queue of EXF or J is released after t_i. If $t_i < e < t^b_{i-1}$, J is scheduled in (t_i, t^b_{i-1}), because EXF never rejects any job in the queue within the interval. Otherwise, *i.e.*, $e \in T_j$, for some $0 \leq j \leq i-1$, then in the case the barrier time t^b_j is determined by an idle time, J is scheduled before t^b_j, because if it were not scheduled until t^b_j, it would be alive in the queue at t^b_j and it would be scheduled. Suppose t^b_j is determined by a job $\hat{J}$ with an expiration time $> t_j$ and J is not scheduled before t^b_j. If $t^b_j \leq e \leq t_j$, it is a contradiction, because $\hat{J}$ has the expiration time greater than J but J would be in the queue at the time when $\hat{J}$ was scheduled. So $t_j < e < t^b_{j-1}$. From Lemma 1, J is not scheduled within the interval $[t^b_j, t_j]$ and it is in queue at t_j. Since EXF never rejects any job in the queue within (t_j, t^b_{j-1}), the job J is scheduled.

If a job J with a processing time ℓ starts to process at s and $s \leq t < s+\ell$, the job J is said to cross the time t. For each principal period T_i, we define *frontier jobs* of T_i in EXF (OPT) as the jobs scheduled by EXF (OPT, resp.) to cross the time $t_i - \lfloor t_i - t^b_i \rfloor$, denoted by f_i. In particular, $\mathcal{F}(T)$ represents the set of all frontier jobs of a principal period T in EXF. Here we define new periods $\{\widetilde{T}_i\}$

from the principal periods $\{T_i\}$. In nondecreasing order, two adjacent principal periods may be merged into one satisfying the following criterion. Assume new periods $\widetilde{T}_k, k = 0, \cdots, j-1$, are defined from $T_k, k = 0, \cdots, i-1$. Let $g_{\widetilde{k}}$ be the time $t^b_k + \lfloor t_k - t^b_k \rfloor + 1$. First set $\widetilde{T}_j$ to be T_i. If $g_{i+1} > f_i$, then set $\widetilde{T}_j$ to be $\widetilde{T}_j \cup T_{i+1}$, that is, two principal periods T_i and T_{i+1} are merged, and otherwise, set $\widetilde{T}_{j+1}$ to be T_{i+1}. Thus the new periods $\{\widetilde{T}_i\}$ are obtained, each of which consists of either one principal period or two principal periods. We will refer to *the sole period* as the period which consists of one principal period and *the combined period* as the period into which two principal periods are merged.

In the following lemmas, we will bound $|EXF(\widetilde{T}_i)|$ and $|OPT(\widetilde{T}_i)|$ for each i.

Lemma 3. *For each sole period* $\widetilde{T}_i$,

$$|EXF(\widetilde{T}_i)| \geq |\mathcal{F}(T_{h_i}) \cap EXF(\widetilde{T}_i)| + m\lfloor t_{h_i} - t^b_{h_i} \rfloor + |\mathcal{L}_{EXF}(T_{h_i})|,$$

where $\widetilde{T}_i$ *is the principal period* T_{h_i}.

Proof. Fix a sole period $\widetilde{T}_i$. From Lemma 1, there is no idle time on any machine in $[t^b_{h_i}, t_{h_i}]$ in the schedule of EXF. So EXF can schedule at least $m\lfloor t_{h_i} - t^b_{h_i} \rfloor$ jobs which are not the frontier jobs of T_{h_i} within this interval.

Lemma 4. *For each sole period* $\widetilde{T}_i$,

$$|OPT(\widetilde{T}_i)| \leq m(\lfloor t_{h_i} - t^b_{h_i} \rfloor + 1) + |\mathcal{L}_{OPT}(T_{h_i})|,$$

where $\widetilde{T}_i$ *is the principal period* T_{h_i}.

Lemma 4 is obvious, since any scheduler can allocate at most $m(\lfloor t_{h_i} - t^b_{h_i} \rfloor + 1)$ jobs in $[t^b_{h_i}, t_{h_i}]$, and Lemma 5 can be shown similarly to Lemma 3.

Lemma 5. *For each combined period* $\widetilde{T}_i$ *which consists of two principal periods* T_{h_i} *and* T_{h_i+1},

$$|EXF(\widetilde{T}_i)| \geq |\mathcal{F}(T_{h_i+1}) \cap EXF(\widetilde{T}_i)| + m(\lfloor t_{h_i} - t^b_{h_i} \rfloor + \lfloor t_{h_i+1} - t^b_{h_i+1} \rfloor) + \sum_{j=h_i}^{h_i+1} |\mathcal{L}_{EXF}(T_j)|.$$

Lemma 6. *For each combined period* $\widetilde{T}_i$ *which consists of two principal periods* T_{h_i} *and* T_{h_i+1},

$$|OPT(\widetilde{T}_i)| \leq m(\lfloor t_{h_i} - t^b_{h_i} \rfloor + \lfloor t_{h_i+1} - t^b_{h_i+1} \rfloor + 1) + |\mathcal{L}_{OPT}(T_{h_i})|.$$

Proof. Consider frontier jobs of T_{h_i} in OPT. Then there are at most $m\lfloor t_{h_i} - t^b_{h_i} \rfloor$ jobs which are not the fronter jobs of T_{h_i} and are scheduled by OPT in $[t^b_{h_i}, t_{h_i}]$. Also since $g_{h_i+1} > f_{h_i}$, at most $m\lfloor t_{h_i+1} - t^b_{h_i+1} \rfloor$ jobs which are not the fronter jobs of T_{h_i} can be scheduled by OPT in $[t^b_{h_i+1}, t^b_{h_i})$. Thus since there can be at most m frontier jobs of T_{h_i}, the upper bound is obtained.

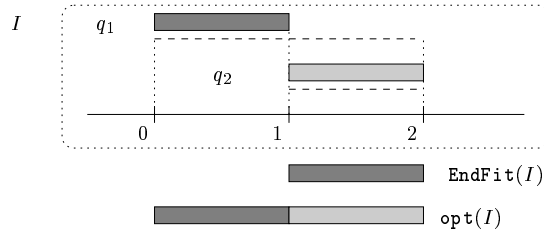


Fig. 2. The schedule of EXF in the case that $\tilde{T}_j$ is the sole period.

From Lemma 2 $\sim$ 6, we will show the performance of EXF when $\lfloor\kappa\rfloor \geq 1$.

Theorem 1. *The EXF algorithm is* $(1 + \frac{1}{\lfloor\kappa\rfloor+1})$*-competitive, where* $\lfloor\kappa\rfloor \geq 1$.

Proof. From Lemma 2, all late jobs of OPT also are late jobs of EXF, that is, for each sole or combined period $\widetilde{T}_i, i = 0, \cdots, K$,

$$\sum_i |\mathcal{L}_{opt}(T_{h_i})| \leq \sum_j |\mathcal{L}_{ext}(T_j)|, \tag{1}$$

where the sole or combined period $\widetilde{T}_i$ consists of the principal period T_{h_i} or the principal periods T_{h_i} and T_{h_i+1}, respectively, and the right sum is done over all principal periods T_j. For convenience, assume that $\widetilde{T}_K$ is the combined period and there is neither an idle time on some machine nor a job with an expiration time $> t_{h_K+1}$ scheduled in $[0, t_{h_K+1}]$. Then $|\mathcal{F}(T_{h_K+1}) \cap EXF(\widetilde{T}_K)| = m$. Thus we obtain the following from Equation (1) and Lemma 3 $\sim$ 6:

$$OPT(\mathcal{J}) \leq EXF(\mathcal{J}) + Km - \sum_{i=0}^{K-1} |\mathcal{F}(T_{k_i}) \cap EXF(\widetilde{T}_i)|,$$

where we define $k_i = h_i$ and $k_i = h_i+1$ for each sole period and combined period $\widetilde{T}_i$, respectively. Also the lower bound of $EXF(\mathcal{J})$ is given by the following claim.

Claim. $EXF(\mathcal{J}) \geq Km\lfloor\kappa\rfloor + Km - \sum_{i=0}^{K-1} |\mathcal{F}(T_{k_i}) \cap EXF(\widetilde{T}_i)|$.

Let $\mathcal{A}_i = \mathcal{F}(T_{k_i}) - EXF(\widetilde{T}_i)$ and $\mathcal{B}_i = EXF([t^b_{k_i}, f_{h_i}]) \cup \mathcal{L}_{EXF}(T_{h_i})$. Then $\mathcal{A}_i$ consists of the jobs which are frontier jobs of T_{k_i} and not scheduled within T_{k_i}, and if $\widetilde{T}_i$ is a combined period, $\mathcal{B}_i$ contains all jobs in $EXF(T_{k_i})$, all frontier jobs of T_{h_i}, and all late jobs of EXF in T_{h_i} and if $\widetilde{T}_i$ is a sole period, it contains the frontier jobs of T_{h_i} belonging to $EXF(T_{h_i})$ and all late jobs of EXF in T_{h_i}. First we can see from Lemma 1 that for each $i = 0, \cdots, K-1$, $|EXF(\widetilde{T}_i) - \mathcal{B}_i| \geq m\lfloor t_{h_i} - t^b_{h_i}\rfloor$. Let $\mathcal{A} = \cup_{i=0}^{K-1}\mathcal{A}_i$ and $\mathcal{B} = \cup_{i=0}^{K}\mathcal{B}_i$. We define a mapping $\mathcal{I} : \mathcal{A} \longrightarrow \mathcal{B}$ as follows. Given $J \in \mathcal{A}$. Then $J \in \mathcal{F}(T_{k_j}) - EXF(\widetilde{T}_j)$, for some j. If J is the late job of EXF in $T_{h_{j+1}}$, that is, $J \in EXF((t_{h_{j+1}}, t^b_{k_j}))$, then set $\mathcal{I}(J) = J$, and if $J \in EXF([t^b_{h_{j+1}}, t_{h_{j+1}}])$, then in the case that $\widetilde{T}_j$ is the combined period, there is a

job $J' \in EXF([t^b_{k_j}, t_{k_j}))$ on the same machine as J, because $\lfloor t_{k_j} - t^b_{k_j} \rfloor \geq \lfloor \kappa \rfloor \geq 1$. For convenience, we choose the job J' scheduled next to J and set $\mathcal{I}(J) = J'$. In the case that $\widetilde{T}_j$ is the sole period, there is a frontier job $J' \in EXF([t^b_{h_{j+1}}, f_{h_{j+1}}])$ on the same machine as J, because otherwise, if the last job in $EXF([t^b_{h_{j+1}}, t_{h_{j+1}}])$ on the machine is completed at $\tilde{t}$, $\tilde{t} \leq g_{h_{j+1}} \leq f_{k_j}$ since $\widetilde{T}_j$ is the sole period, that is, J cannot be the frontier job of T_{k_j} and it is a contradiction (see Figure 2). Then we set $\mathcal{I}(J) = J'$. It is obvious that the mapping $\mathcal{I}$ is one to one, which says $|\mathcal{A}| \leq |\mathcal{B}|$. Let $\mathcal{E}_i = \mathcal{F}(T_{k_i}) \cap EXF(\widetilde{T}_i)$. Then since $|\mathcal{A}| = Km - \sum_{i=0}^{K-1} |\mathcal{E}_i|$, we can prove the claim from the following:

$$\begin{aligned} EXF(\mathcal{J}) &\geq \sum_{i=0}^{K-1} |EXF(\widetilde{T}_i) - \mathcal{B}_i| + |\mathcal{B}| \\ &\geq m \sum_{i=0}^{K-1} \lfloor t_{h_i} - t^b_{h_i} \rfloor + |\mathcal{A}| \\ &\geq Km\lfloor \kappa \rfloor + Km - \sum_{i=0}^{K-1} |\mathcal{E}_i|. \end{aligned}$$

Consequently, we obtain the desired bound:

$$\frac{OPT(\mathcal{J})}{EXF(\mathcal{J})} \leq 1 + \frac{Km - \sum_i |\mathcal{E}_i|}{Km\lfloor \kappa \rfloor + Km - \sum_i |\mathcal{E}_i|} \leq 1 + \frac{1}{\lfloor \kappa \rfloor + 1}.$$

2.2 Case $\lfloor \kappa \rfloor = 0$

In this subsection, when $\lfloor \kappa \rfloor = 0$, it is shown that the EXF algorithm is 2-competitive. We obtain the set of the principal periods $\{T_i\}$ as in the previous and merge them differently. New periods $\widetilde{T}_i$ are obtained as follows: Assume $\widetilde{T}_k, k = 0, \cdots, j-1$, are obtained from $T_k, k = 0, \cdots, i-1$. If there is a job which starts to process in $[t^b_l, t_l]$ and is completed after t_i, for some $l > i$, then let T_u be the principal period having the largest index $u > i$ in which such a job exists, and we merge the principal periods $T_i, \cdots, T_u$ into one as $\widetilde{T}_j$. Otherwise, set $\widetilde{T}_j$ to be T_i. Thus the periods $\widetilde{T}_i$ which consist of one principal period and principal periods more than one are called *long periods* and *short periods*, respectively. Let $\mathcal{J}_i$ and $\widetilde{\mathcal{J}}_i$ be the set of all jobs which is released within T_i and $\widetilde{T}_i$, respectively. Then we compare $EXF(\widetilde{\mathcal{J}}_i)$ with $OPT(\widetilde{\mathcal{J}}_i)$ for each i. In each $\mathcal{J}_i$, the jobs having expiration times $> t_i$ are called *patient jobs* and it can be shown similarly to Lemma 2 that all patient jobs should be scheduled by EXF. Also we note that in each T_i, all jobs scheduled by EXF in $[t^b_i, t_i]$ belong to $\mathcal{J}_i$.

Lemma 7. *In each principal period T_i, all jobs scheduled by EXF in $[t^b_i, t_i]$ belong to $\mathcal{J}_i$, that is, they are released after or at the barrier time t^b_i of T_i, and all patient jobs of $\mathcal{J}_i$ must be scheduled by EXF.*

Lemma 8. *For each long period* $\widetilde{T}_i$,

$$OPT(\widetilde{\mathcal{J}}_i) \leq EXF(\widetilde{\mathcal{J}}_i) + m.$$

Lemma 9. *For each short period* $\widetilde{T}_i$,

$$OPT(\widetilde{\mathcal{J}}_i) \leq EXF(\widetilde{\mathcal{J}}_i) + m.$$

Proof. Fix a short period $\widetilde{T}_i$ which consists of principal periods $T_j, \cdots, T_k$. If $g_k = t_k^b + \lfloor t_k - t_k^b \rfloor + 1 \leq t_j$, then since $\widetilde{T}_i$ is the short period, we see that $t_j - t_k < 1$. Thus OPT can schedule at most $m(\lfloor t_k - t_k^b \rfloor + 2)$ jobs in $[t_k^b, t_j]$. So the upper bound of $OPT(\widetilde{\mathcal{J}}_i)$ is given by

$$OPT(\widetilde{\mathcal{J}}_i) \leq m(\lfloor t_k - t_k^b \rfloor + 2) + |\mathcal{P}_j|,$$

where $\mathcal{P}_j$ is the set of all patient jobs of $\mathcal{J}_j$. But EXF schedules at least $m\lfloor t_k - t_k^b \rfloor$ jobs which are not the frontier jobs of T_k in $[t_k^b, t_k]$. Also the m frontier jobs of T_j exist, and if some of them start to process before or at t_k, there is the frontier job of T_k on the same machine, because $g_k \leq t_j$. Thus we can obtain the m jobs of $\widetilde{\mathcal{J}}_i$ each of which is either a frontier job of T_j scheduled after t_k or a frontier job of T_k. Therefore the following is given from Lemma 7:

$$EXF(\widetilde{\mathcal{J}}_i) \geq m(\lfloor t_k - t_k^b \rfloor + 1) + |\mathcal{P}_j|.$$

If $g_k > t_j$, then OPT can schedule at most $m(\lfloor t_k - t_k^b \rfloor + 1)$ jobs in $[t_k^b, t_j]$ but EXF schedules at least $m\lfloor t_k - t_k^b \rfloor$ jobs in $[t_k^b, t_k]$. Thus the desired equation is obtained.

Theorem 2. *The EXF algorithm is 2-competitive, where* $\lfloor \kappa \rfloor = 0$.

Proof. Given the long or short periods $\widetilde{T}_i, i = 0, \cdots, K-1$. Actually it remains to prove $\sum_i EXF(\widetilde{\mathcal{J}}_i) \geq Km$. Fix $\widetilde{T}_i$ and we assume that $\widetilde{T}_i$ consists of the principal periods $T_j, \cdots, T_{j+k}$, where $k \geq 0$, and $k = 0$ if $\widetilde{T}_i$ is the long period. Then from the definition of the critical time t_j we can see that there are exactly m jobs crossing t_j. Thus we assign the m jobs to $\widetilde{T}_i$. If $\widetilde{T}_i$ is the long period, any job scheduled before or at t_{j+1} cannot be completed after t_j, and if $\widetilde{T}_i$ is the short period, any job scheduled before or at t_{j+k+1} cannot be completed after t_j, because if there is such a job, it is a contradiction to the maximality of T_{j+k} in the definition of $\widetilde{T}_i$. Thus we can see that the m jobs assigned to $\widetilde{T}_i$ cannot be re-assigned by any other periods and the desired fact is proved. Consequently, it is followed from Lemma 8 and 9 that

$$OPT(\mathcal{J}) \leq EXF(\mathcal{J}) + Km \leq 2EXF(\mathcal{J}).$$

Until now, we have proved that EXF is $(1 + \frac{1}{\lfloor \kappa \rfloor + 1})$-competitive, for $\kappa \geq 0$. Next, it is shown that the competitive ratio is tight by giving the lower bound. The instance of the lower bound is given just by making m copies for each job in the instance given in [8].

Lemma 10. *For $\kappa \geq 0$,*

$$\max_{\mathcal{J}} \frac{OPT(\mathcal{J})}{EXF(\mathcal{J})} \geq 1 + \frac{1}{\lfloor \kappa \rfloor + 1}$$

and, for any online algorithm A,

$$\max_{\mathcal{J}} \frac{OPT(\mathcal{J})}{A(\mathcal{J})} \geq 1 + \frac{1}{2\lfloor \kappa \rfloor + 3}.$$

3 Arbitrary Processing Time Jobs

As in the last section, the principal periods T_i are defined, and the barrier times t_i^b and the critical times t_i which determine T_i are given. Let $\mathcal{J}_i$ be the set of all jobs which are released within T_i. It can be shown similarly to Lemma 7 that all patient jobs of $\mathcal{J}_i$ having expiration times $> t_i$ should be scheduled by EXF.

Lemma 11. *In each principal period T_i, there is no idle time on any machine between the barrier time t_i^b and the critical time t_i, and all patient jobs of $\mathcal{J}_i$ must be scheduled by EXF. Thus we get*

$$EXF(\mathcal{J}) \geq \max\{\sum_i m\lfloor t_i - t_i^b \rfloor, \sum_i \sum_{j \in \mathcal{P}_i} p_j\},$$

where $\mathcal{P}_i$ is the set of all patient jobs of $\mathcal{J}_i$.

Lemma 12. *For each principal period T_i,*

$$OPT(\mathcal{J}_i) \leq (1 + \frac{1}{\kappa})m\lfloor t_i - t_i^b \rfloor + \sum_{j \in \mathcal{P}_i} p_j,$$

where $\mathcal{P}_i$ is the set of all patient jobs of $\mathcal{J}_i$.

Proof. Let J be a job of $\mathcal{J}_i$ scheduled to cross the time t_i by OPT. In fact, there can be at most m jobs crossing t_i. If the job J is not the patient job of $\mathcal{J}_i$, its slack time is no more than $\lfloor t_i - t_i^b \rfloor$. From the slackness condition, the processing time ℓ of J is less than or equal to the slack time divided by κ, that is, $\ell \leq \frac{\lfloor t_i - t_i^b \rfloor}{\kappa}$. Futhermore, the total gain of all the jobs in $\mathcal{J}_i$ which can be scheduled to be completed before t_i is at most $m\lfloor t_i - t_i^b \rfloor$. Thus we obtain the bound.

Now, we prove that EXF is $(2 + \frac{1}{\kappa})$-competitive, for $\kappa > 0$, and we can also obtain the instance of the lower bound similarly to in [8].

Theorem 3. *The EXF algorithm is $(2 + \frac{1}{\kappa})$-competitive, where $\kappa > 0$.*

Proof. From Lemma 11 and 12, it is shown that

$$OPT(\mathcal{J}) \leq \sum_i (1+\frac{1}{\kappa}) m \lfloor t_i - t_i^b \rfloor + \sum_i \sum_{j \in \mathcal{P}_i} p_j$$
$$\leq (1+\frac{1}{\kappa}) EXF(\mathcal{J}) + EXF(\mathcal{J}) = (2+\frac{1}{\kappa}) EXF(\mathcal{J}).$$

Lemma 13. *For $\kappa > 0$,*

$$\max_{\mathcal{J}} \frac{OPT(\mathcal{J})}{EXF(\mathcal{J})} \geq 2 + \frac{1}{\kappa}.$$

References

[1] A. Bar-Noy, R. Bar-Yehuda, J. S. Naor A. Freud, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *Proc. of the 32nd Annual ACM Symposium on Theory of Computing*, pages 735–744, 2000.

[2] A. Bar-Noy, J.S. Naor S. Guha, and B. Schieber. Approximating the throughput of multiple machines under real-time scheduling. In *Proc. of the 31st Annual ACM Symposium on Theory of Computing*, pages 622–631, 1999.

[3] S. Baruah and J. R. Haritsa. Scheduling for overload in real-time systems. *IEEE Transactions on Computers*, 46:1034–1039, 1997.

[4] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha. On-line scheduling in the presence of overload. In *Proc. of the 32nd Annual Symposium on Foundation of Computer Science*, pages 100–110, 1991.

[5] Bhaskar DasGupta and Michael A. Palis. Online real-time preemptive scheduling of jobs with deadlines. In *3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 1913 of *Lecture Notes in Computer Science (LNCS)*, pages 96–107. Springer-Verlag, 2000.

[6] M. R. Garey and D. S. Johnson. Two processor scheduling with start times and deadlines. *SIAM J. on Computing*, 6:416–426, 1977.

[7] S. Goldman, J. Parwatikar, and S. Suri. On-line scheduling with hard deadlines. In *Proc. of the Workshop on Algorithms and Data Structures (WADS)*, volume 1272 of *Lecture Notes in Computer Science (LNCS)*, pages 258–271. Springer-Verlag, 1997.

[8] Micheal H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. In *Proc. of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 396–405, 1999.

[9] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy paging. *Algorithmica*, 3:70–119, 1998.

[10] R. Lipton and A. Tomkins. Online interval scheduling. In *Proc. of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.

[11] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23:116–127, 1976.

[12] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

Competitive Online Scheduling with Level of Service*

(Extended Abstract)

Ee-Chien Chang[1] and Chee Yap[2]

[1] Department of Computational Science
National University of Singapore
changec@cz3.nus.edu.sg
[2] Department of Computer Science
Courant Institute, New York University

Abstract. Motivated by an application in thinwire visualization, we study an abstract on-line scheduling problem. Unlike most scheduling problems, our schedulers can gain partial merit from a partially served request. Thus our problem embodies a notion of "Level of Service" that is increasingly important in multimedia applications. We give two schedulers `FirstFit` and `EndFit` based on two simple heuristics, and generalize them into a class of greedy schedulers. We show that both `FirstFit` and `EndFit` are 2-competitive, and any greedy scheduler is 3-competitive. These bounds are shown to be tight.

1 Introduction

We study an abstract on-line scheduling problem motivated by visualization across a "thinwire" network [4, 3]. An example of such a visualization problem is a server-client model where the server and client are connected by a thinwire (that is, a bandwidth-limited connection such as the Internet), with the server holding a very large image that the client wishes to visualize. The viewer on the client side can control the transmission process by moving a mouse cursor over a low-resolution copy of the image to be visualized. This mouse motion generates, in real-time, a sequence of sampled positions along the mouse cursor trajectory. Each sampled position (x, y) corresponds to a request for higher resolution data at the position. As the bandwidth is limited, we could only partially serve each request. This is where an on-line scheduler is needed to optimize the decisions. In most scheduling problems, a partially served request does not contribute to the performance of the scheduler. However, in this problem, a partially sent data can still provide useful information to the user. Thus, instead of sending all the requested data, the server has the option of lowering the "level" of the requested service in order to gain an overall better response time. This paper focuses on this level of service property. Note that there is considerable interest in similar Quality of Service (QoS) issues in multimedia research.

* This research was partially funded by NSF grant CCR-9619846.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 453–462, 2001.
© Springer-Verlag Berlin Heidelberg 2001

We use the standard notion of "competitiveness" in the sense of Sleator and Tarjan [8]. to judge the quality of our online schedulers. A *scheduler* S produces a feasible schedule $S(I)$ for each instance I of our scheduling problem. Each $S(I)$ has an associated *merit*, where $\mathit{merit}(S(I)) \geq 0$. Let $\mathtt{opt}(I)$ denote any feasible schedule for I that maximizes the merit. We say S is *c-competitive* ($c \geq 1$) if for all I,

$$\mathit{merit}(\mathtt{opt}(I)) \leq c \cdot \mathit{merit}(S(I)) + b,$$

where b is a fixed constant. The *competitive ratio* of S is defined by

$$C(S) := \sup_I \frac{\mathit{merit}(\mathtt{opt}(I))}{\mathit{merit}(S(I))}.$$

Thus, we want schedulers S with $C(S) \geq 1$ as small as possible. There is a fairly large literature on competitive algorithms (e.g., [1, 2, 7]. The class of problems most closely related to ours is the online interval packing problem for a single server, where a schedule is a subset of non-overlapping intervals. Lipton and Tomkins [6] study a variant where the input intervals are sorted by their left endpoints. They give a randomized scheduler that is 2-competitive. As we will see, our problem is different from theirs in several ways. Woeginger [9] studied a problem that has several of the features of our problem. Other online interval packing problems can be found in [10, 5].

2 Problem Formulation

We formalize our problem as an on-line scheduling problem. Each *request* q has four parameters

$$q = (s, t, v, w),$$

where s the *start time*, t the *termination time* (or deadline), v is the *volume* (or size), and w is the *weight*. We require

$$v \geq 0 \quad \text{and} \quad w \geq 0.$$

Write $\mathtt{st}(q)$, $\mathtt{dl}(q)$, $\mathtt{sz}(q)$, $\mathtt{wt}(q)$ for the above parameters of q, respectively. Call the half-open interval $(s, t]$ the *span* of q, written $\mathit{span}(q)$. A request q can only be served within its span $(s, t]$, and at any time moment t_0, at most one request can be served.

An *instance* I is a sequence $\langle q_1, q_2 \ldots q_n \rangle$ of requests where the start times of the q_i's are in increasing order: note that we allow $\mathtt{st}(q_i) = \mathtt{st}(q_{i+1})$ even though we nominally say q_i starts before q_{i+1}. How requests are served is described by the schedule. Formally, a *schedule* for I is a piece-wise constant function

$$H : \mathbb{R} \to \{q_1, q_2, \ldots, q_n\} \cup \{\emptyset\},$$

where $|H^{-1}(q_k)| \leq \mathtt{sz}(q_k)$ and $H^{-1}(q_k) \subseteq \mathit{span}(q_k)$ for $k = 1, \ldots, n$. Intuitively, $H(t_0) = q_k$ means the kth request is served at time t_0 and $H(t_0) = \emptyset$ means

no request is being served. A time moment t_0 is called a *breakpoint* if H is discontinuous at t_0. (More precisely, for every $\varepsilon > 0$, there exists δ_i $(0 < \delta_i < \varepsilon$, $i = 1, 2)$ such that $H(t_0) = H(t_0 - \delta_1) \neq H(t_0 + \delta_2))$. We further require a schedule H to has finitely many breakpoints. In addition, for each request q, $|H^{-1}(q)| \leq \mathtt{sz}(q)$. A half-open interval of the form $(t_0, t_1]$ is called a *time-slot*. Without loss of generality, we may assume $H^{-1}(q)$ is a finite union of time slots. The merit *merit*(H) of a schedule is

$$\sum_{j=1}^{n} \mathtt{wt}(q_j)|H^{-1}(q_j)|.$$

Relative to a schedule H at any time t_0, we call

$$v' := \mathtt{sz}(q) - |H^{-1}(q) \cap (-\infty, t_0]|$$

the *residual size* of q. A request q is completely served if $v' = 0$. If $v' > 0$ and $\mathtt{st}(q) \leq t_0 \leq \mathtt{dl}(q)$, then we say q is *pending*. The *residue* of a pending q at time t_0 is the modified request $q' = (t_0, \mathtt{dl}(q), v', \mathtt{wt}(q))$.

For each completely served request, the scheduler gains $\mathtt{sz}(q)\mathtt{wt}(q)$ merit points (so weights are multiplicative). Moreover, partially served requests gains a proportional fraction of this merit. This is unlike usual scheduling problems in which partially served requests receive no merit. Our model is meaningful for the "foveated visualization" studied in [4, 3] because a scheduler can reduce the amount of requested visualization data along a finely graduated scale. This can be achieved by reducing two parameters, the *foveal radius* and/or *foveal resolution*. The foveal radius measures how fast the resolution falls away from the foveal center, and the foveal resolution measures the maximum resolution (which is at the foveal center).

Preemption. It is implicit in the above definitions that the servicing of any request can be preempted as often as we like with no penalty. Hence we may imagine the scheduler to "plan" a schedule based on all the currently residual requests. It services the requests according to this plan until the arrival of a new request. Then it suspends the current plan, recomputes a new plan based on the new set of residual requests, and repeats the process.

Optimal Schedules. We say H is *optimal* for I if *merit*(H) is maximum among all schedules for I. The existence of optimal schedules is not immediate.

Lemma 1. *For all sequences I of n requests, there exists an optimal schedule with at most $2n^2 + n$ breakpoints.*

The optimal schedules for an instance may not be unique. In the full paper, we give a canonical representation whereby the optimal schedule is unique. Let $\mathtt{opt}(I)$ denote the canonical optimal schedule for I.

Ordering of Requests. The schedulers in this paper make decisions by giving priority to heavier weighted requests. In case $wt(p) = wt(q)$, we resolve the tie by treating p as "heavier" than q if and only if p starts before q.

2.1 FirstFit

The online scheduler that always serves the heaviest residual request at each moment is called the FirstFit scheduler. Figure 1 shows the schedule produced by FirstFit on an instance of two requests q_1 and q_2. Although this example may appear contrived, we can modify q_1 to $\widetilde{q}_1$ where $\widetilde{q}_1 = (0, 2, 1, 1+\epsilon)$. For any $\epsilon > 0$, the FirstFit schedule is the one shown in Figure 1.

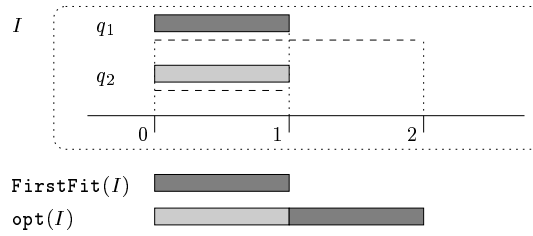


Fig. 1. The top figure illustrates the instance $I = \langle q_1, q_2 \rangle$ where $q_1 = (0, 2, 1, 1)$ and $q_2 = (0, 1, 1, 1)$. Each horizontal "dashed" line represents the span of the request. Although $\mathtt{wt}(q_1) = \mathtt{wt}(q_2)$, q_1 is "heavier" than q_2 by our tie-breaking rule. In $\mathtt{opt}(I)$, q_1 and q_2 are served in the time-slots $(1, 2]$ and $(0, 1]$, respectively. However, in $\mathtt{FirstFit}(I)$, only q_1 is served.

2.2 EndFit

Consider an online scheduler which always serves according to the optimal schedule for the current set of residual requests. This was first suggested by Estie Arkin[1]. To implement such a scheduler, we can invoke a general off-line algorithm for computing optimal schedules upon each new arrival of a request. But it turns out that a very simple scheduler can be used. This scheduler, on an arbitrary instance I, operates as follows:

> Starting from the heaviest request down to the lightest, allocates each request $q \in I$ to the latest available time-slot(s) within $span(q)$.

Call this the OffEndFit scheduler. It is an off-line algorithm because it must see the entire set of requests to make its decisions. This scheduler is optimal for a special class of instances.

Lemma 2. *If I is an instance in which all requests have a common starting time, then $\mathtt{OffEndFit}(I)$ is the canonical optimal schedule for I.*

Let EndFit be the online scheduler which always serves according to the OffEndFit schedule for the residual requests. More precisely, on arrival of a new

[1] Private communication (1997).

request q, `EndFit` preempts the current service. It computes a new schedule P for the current residual requests using `OffEndFit`, and continues by servicing P. Call P the *plan* upon arrival of q. Since all residual requests have a common starting time, P is the canonical optimal schedule. Figure 2 shows the `EndFit` schedule for an instance $I = (q_1, q_2)$. The `EndFit` schedule for this example may appear contrived. To see that it is "correct in the limit", let $I_\epsilon = \langle q_0, q_1, q_2 \rangle$ where $q_0 = (0, 1, 1, \epsilon)$. For any $0 < \epsilon < 1$, the off-line optimal schedule for the current residual requests is unique. As $\epsilon \to 0$, $\texttt{EndFit}(I_\epsilon)$ approaches the one schedule shown in Figure 2.

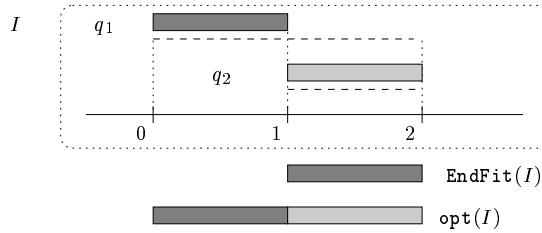


Fig. 2. The top figure illustrates the instance I of two requests: $q_1 = (0, 2, 1, 1)$ and $q_2 = (1, 2, 1, 1)$. In the $\texttt{opt}(I)$, q_1 and q_2 are served. However, in $\texttt{EndFit}(I)$, only q_1 is served.

3 Competitive Ratio of `FirstFit`

Example 1 (Figure 1) shows that the competitive ratio of `FirstFit` is at least 2. We will show that `FirstFit` is 2-competitive. Before presenting the proof, let us give two definitions.

Charging Scheme. Let H, H_1 and H_2 be schedules for an instance I. We often need to argue that the merit of H is no larger than the sum of the merits of H_1 and H_2. Our approach is to *charge* a portion of H to H_1 and the remaining to H_2. Intuitively, the charging process can be viewed as first cutting H_1 and H_2 into pieces and then piecing them together again to form another piecewise-constant function $H_{\texttt{chg}}$. Each piece, after cutting, may be translated before being placed into their slot in $H_{\texttt{chg}}$. As it may turn out that $|H_{\texttt{chg}}^{-1}(q)| > \texttt{sz}(q)$ for some request q, $H_{\texttt{chg}}$ is not necessarily a schedule. The cut-and-paste is done in a way that for all t, $\texttt{wt}(H_{\texttt{chg}}(t)) \geq \texttt{wt}(H(t))$. Therefore,

$$merit(H) \leq merit(H_{\texttt{chg}}) \leq merit(H_1) + merit(H_2). \tag{1}$$

In particular, if $H_1 = H_2$, then we have $merit(H) \leq 2 \cdot merit(H_1)$. When we use the phrase: "charge $(s', e']$ from H to H_1 at $(s, e]$", we mean that a piece $(s, e]$ is cut from H_1 and placed into the slot $(s', e']$ in $H_{\texttt{chg}}$. Equivalently, if $H^{-1}(q)$

is the interval $(s', e']$, we may say that the request q is charged from H to H_1 at $(s, e]$ (see Figure 3). Thus, to show (1), we need to charge each time slot of H to either H_1 or H_2, and ensure that no part of H_1 or H_2 is charged more than one.

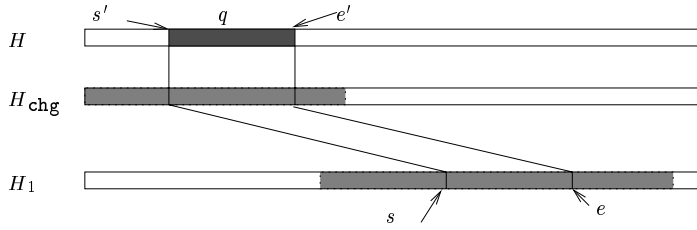


Fig. 3. Charging the request q in H to H_1 at $(s, e]$.

Intactness. A request q is *intact* in a schedule H if $H^{-1}(q)$ is connected, and either $|H^{-1}(q)| = 0$ or $\mathtt{sz}(q)$. Most of our proofs will be simplified if we assume intactness.

Theorem 1. *For any instance I,*

$$\text{merit}(\mathtt{opt}(I)) \leq 2 \cdot \text{merit}(\mathtt{FirstFit}(I)).$$

Proof. Given an instance I, let $H_{\mathtt{ff}} := \mathtt{FirstFit}(I)$ and $H_{\mathtt{opt}} := \mathtt{opt}(I)$. We can assume that requests in I are intact in both $H_{\mathtt{ff}}$ and $H_{\mathtt{opt}}$.

Let H_0 be an identical copy of $H_{\mathtt{ff}}$. We want to charge requests served in $H_{\mathtt{opt}}$ to H_0 and $H_{\mathtt{ff}}$. Let $\{t_1, t_2, \ldots, t_m\}$ be the distinct breakpoints in $H_{\mathtt{opt}}$, where $t_i < t_j$ if and only if $i < j$.

For each t_i, starting from $i := 1$ to $m - 1$, consider the time-slot $(t_i, t_{i+1}]$. Let $q_{\mathtt{opt}} := H_{\mathtt{opt}}(t_{i+1})$. Let $q_{\mathtt{ff}}$ be the lightest request served during $(t_i, t_{t+1}]$ by `FirstFit`. There are two cases:

1. If the request $q_{\mathtt{ff}}$ is not lighter than $q_{\mathtt{opt}}$, charge $q_{\mathtt{opt}}$ from $H_{\mathtt{opt}}$ to $H_{\mathtt{ff}}$ at $(t_i, t_{i+1}]$.
2. Otherwise, charge $q_{\mathtt{opt}}$ from $H_{\mathtt{opt}}$ to H_0 at $H_0^{-1}(q_{\mathtt{opt}})$.

We have to show that in the second case, $|H_0^{-1}(q_{\mathtt{opt}})| \geq |(t_i, t_{i+1}]|$. In the first place, why is the weight of $q_{\mathtt{ff}}$ lighter? The request $q_{\mathtt{ff}}$ is chosen by `FirstFit` because it is the heaviest request among the pending requests. This implies that $q_{\mathtt{opt}}$ is not a pending request, even though t_i is in the span of $q_{\mathtt{opt}}$. So $q_{\mathtt{opt}}$ must have been completely served by `FirstFit`. This implies that $|H_0^{-1}(q_{\mathtt{opt}})| \geq |(t_i, t_{i+1}]|$.

4 Competitive Ratio of `EndFit`

Example 2 (Figure 2) shows that the competitive ratio of `EndFit` is at least 2. We now show that this constant is the best possible.

The upper bound proof is considerably more subtle than the proof for FirstFit. The key result is Theorem 2 below which formalizes this observation about EndFit: it never hurts the performance of EndFit to have a request started at an earlier time. For example, in Figure 2, the performance of EndFit will improve if the request q_2 starts at an earlier time. The analogous fails for FirstFit. For example, in Figure 1, the performance of FirstFit would improve if q_1 starts at a time later than 0.

A request $\widetilde{q}$ is a *trimmed* version of q if $\mathtt{st}(\widetilde{q}) \geq \mathtt{st}(q)$, $\mathtt{dl}(\widetilde{q}) = \mathtt{dl}(q)$, $\mathtt{wt}(\widetilde{q}) = \mathtt{wt}(\widetilde{q})$ and $\mathtt{sz}(\widetilde{q}) \leq \mathtt{sz}(q)$. Thus, a trimmed version of q may start later than the original q. An instance $\widetilde{I}$ is a trimmed instance of I if there is a one-one (not necessarily onto) mapping from $\widetilde{I}$ to I such that any $\widetilde{q}$ in $\widetilde{I}$ is a trimmed version of its corresponding request in I. Clearly, $\mathit{merit}(\mathtt{opt}(\widetilde{I})) \leq \mathit{merit}(\mathtt{opt}(I))$. Similar relationship also holds for EndFit.

Theorem 2. *If $\widetilde{I}$ is a trimmed version of I then*

$$\mathrm{merit}(\mathtt{EndFit}(\widetilde{I})) \leq \mathrm{merit}(\mathtt{EndFit}(I)).$$

We now use Theorem 2 to show that EndFit is 2-competitive.

Theorem 3. *For any instance I,*

$$\mathrm{merit}(\mathtt{opt}(I)) \leq 2 \cdot \mathrm{merit}(\mathtt{EndFit}(I)).$$

Proof. We can assume that requests in I are intact in $\mathtt{opt}(I)$. Let $\widetilde{I}$ be the trimmed instance of I such that for any request $q \in I$, if $(\mathtt{opt}(I))^{-1}(q) = (t_1, t_2]$, then the corresponding trimmed request $\widetilde{q}$ satisfies $\mathtt{st}(\widetilde{q}) := t_1$ and $\mathtt{sz}(\widetilde{q}) := t_2 - t_1$; otherwise if $\mathtt{opt}(I)^{-1}(q) = \emptyset$, then $\widetilde{q}$ satisfies $\mathtt{sz}(\widetilde{q}) := 0$. We can further assume that $\widetilde{I}$ is intact in all the plans of EndFit with $\widetilde{I}$. (Recall that a plan is the optimal schedule for the residues). By definition, we have

$$\mathit{merit}(\mathtt{opt}(I)) = \mathit{merit}(\mathtt{opt}(\widetilde{I})). \tag{2}$$

The instance $\widetilde{I}$ has the nice property that requests arrive at a "constant rate", that is, if a request q starts at time t, then no other request starts during $(t, t + \mathtt{sz}(q))$. Let H_1 and H_2 be two identical copies of $\mathtt{EndFit}(\widetilde{I})$. Our theorem is proved if we show how to charge requests in $\mathtt{opt}(\widetilde{I})$ to H_1 and H_2.

Consider a request q in $\widetilde{I}$. Let P^+ be the plan upon the arrival of q and P^- be the plan just before the arrival of q. There are two cases.

1. If q is allocated in the new plan P^+, then it is possible that there are some requests which are originally allocated in P^-, but not in the new plan. Call these requests the *ousted requests*.
2. Otherwise, call q the ousted request.

Let s be the total size of the ousted requests. Note that $s \leq \mathtt{sz}(q)$ and the total merit of the ousted requests is not more than the total merit of the requests

allocated in $(\mathtt{st}(q), \mathtt{st}(q) + s]$ in P^+. Furthermore, the new plan will be carried out without interruption at least until $\mathtt{st}(q) + \mathtt{sz}(q)$. Charge the ousted requests to H_2 at $(\mathtt{st}(q), \mathtt{st}(q) + \mathtt{sz}(q)]$ and the served requests during $(\mathtt{st}(q), \mathtt{st}(q) + \mathtt{sz}(q)]$ to H_1 at $(\mathtt{st}(q), \mathtt{st}(q) + \mathtt{sz}(q)]$. The above is a valid charging scheme. Thus, we have

$$2 \cdot \mathit{merit}(\mathtt{EndFit}(\widetilde{I})) \geq \mathit{merit}(\mathtt{opt}(\widetilde{I})).$$

By Theorem 2 and (2), we have

$$2 \cdot \mathit{merit}(\mathtt{EndFit}(I)) \geq \mathit{merit}(\mathtt{opt}(I)).$$

5 A Class of Greedy Schedulers

Looking at the behavior of `EndFit` and `FirstFit` on specific examples, it appears that they are complementary in the sense that if `EndFit` performs poorly on an instance, then `FirstFit` will perform well, and vice-versa. This suggests studying some combination of these two heuristics and motivates the generalization to a class of `Greedy` schedulers.

A scheduler S in this class behaves as follows.

(A) At the moment a new request q starts, it suspends the current service (that is, it preempts the currently served request).

(B) Scheduler S computes a new *plan* H, which is a schedule for the set of residues of currently pending requests. We call H a 'plan' because the scheduler may not carry out the schedule as planned due to the subsequent new requests. The plan H is computed by considering the residues one by one, starting from the heaviest request down to the lightest request. Let p be the request being considered and call $|H^{-1}(p)|$ the allocation to p. The allocation to p is subjected to the following restriction:

($*$) The allocation to p must be maximized. For example, if it is possible to completely allocate p, the whole of p must be allocated. However, there is no restriction on where p is allocated. Time-slots, once allocated, are not subsequently revised in creating this plan.

(C) It carries out the plan until a new request starts, whereupon we go back to step (A).

Different members of the `Greedy` class differ only in their strategies for (B) subjected to the restriction($*$). Note that our first example `FirstFit` is a `Greedy` scheduler: the request p in ($*$) is allocated in the earliest possible time-slots. The second example `EndFit` is also a greedy scheduler. Its strategy for (B) is rather counter-intuitive: the request p is allocated in the *latest* possible time slots.

By combining the counter examples for `FirstFit` and `EndFit`, we can find a `Greedy` scheduler whose competitive ratio ≥ 3. The next theorem shows that this bound of 3 cannot be improved.

Theorem 4. *Every* `Greedy` *scheduler is 3-competitive.*

6 General Lower Bound

From the previous section, we know that every greedy scheduler is 3-competitive. Are there schedulers outside the `Greedy` class with competitive ratio less than 2? We note a partial result in this direction: *every deterministic scheduler has competitive ratio at least* $2(2-\sqrt{2}) > 1.17$.

In proof, consider this adversary: at time 0, the adversary releases two requests $q_0 := (0, 2, 1, 1)$ and $q_1 := (0, 1, 1, \sqrt{2}-1)$. At time $t = 1$, let the residual size of q_0 be s_0. If s_0 is less than $1/2$ then the request $q_2 := (1, 2, 1, 1)$ is released. Otherwise, no further requests will be released. It may be verified that any deterministic scheduler achieve a merit of at most $\frac{1}{2(2-\sqrt{2})}$ of the maximum possible.

Unfortunately, this lower bound of 1.17 leaves a wide gap from the current upper bound of 2. On the other hand, no simple variation of this adversary seems to give a better lower bound.

7 On the Number of Breakpoints and Multi-tasking

Both `FirstFit` and `EndFit` make $O(n)$ breakpoints where n is the number of requests. Does the number of breakpoints affect the performance of a scheduler? We give an alternative formulation of the scheduling problem which could be viewed as allowing an infinite number breakpoints. In this *multi-tasking* environment, several requests can be served concurrently, but each at a (possibly) different rate. However, in any time interval of size Δt, the total size of requests served within this interval must not exceed Δt. Equivalently, we allow "fractional service" where the total service at any moment sums to 1.

A concrete example of such a scheduler is `FirstEndFit`: It simulates `FirstFit` and `EndFit` concurrently and serves half of what `FirstFit` and `EndFit` would serve. That is, if `FirstFit` and `EndFit`will serve q and p respectively in the time slot $(s_0, t_0]$, then `FirstEndFit` will serve p and q concurrently but each at half the rate. We suspect that `FirstEndFit` is (3/2)-competitive.

Note that `FirstEndFit` can be also viewed as the following randomized scheduler under the original single-tasking setting: Before receiving any request, it tosses a fair coin; If the outcome is `head`, then it simulates `FirstFit`, otherwise it simulates `EndFit`.

Clearly, the expected merit gained by this randomized scheduler is same as the merit gained by `FirstEndFit`.

8 Conclusion

We have formulated a "level-of-service" scheduling problem that arises naturally in our thinwire visualization applications. This formulation is also useful in real-time systems where quality of jobs can be traded-off for time. We have derived several competitive algorithms in this setting. We continue to investigate the

many interesting questions that are open. Besides sharpening the results in the paper, we pose the following directions for further work: (1) Find optimal schedulers which are not restricted to be on-line. (2) Study the problem with other measures of merit (instead of multiplicative weights in this paper). (3) Introduce a model of penalty for preemption. (4) Introduce randomization.

Acknowledgments

We thank Estie Arkin and Yi-Jen Chiang for discussions about the problem.

References

1. S. Albers. Competitive online algorithms. BRICS Lecture Series LS-96-2, BRICS, Department of Computer Science, University of Aarhus, September 1996.
2. S. Albers and J. Westbrook. A survey of self-organizing data structures. Research Report MPI-I-96-1-026, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, October 1996.
3. E. Chang, C. Yap, and T.-J. Yen. Realtime visualization of large images over a thinwire. In *IEEE Visualization '97 (Late Breaking Hot Topics)*, pages 45–48, 1997.
4. E.-C. Chang. *Foveation Techniques and Scheduling Issues in Thinwire Visualization.* PhD thesis, Department of Computer Science, New York University, May 1998.
5. J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. *Integer Programming and Combinatorial Opt.*, pages 404–414, 1996.
6. R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 302–311, 1994.
7. L. A. McGeoch and D. D. Sleator, editors. *On-Line Algorithms.* DIMACS series in Discrete Mathematics and Theoretical Computer Science, volume 7. AMS, 1992.
8. D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Comm. of the ACM*, 28(2):202–208, 1985.
9. G. J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Computer Science*, 130:5–16, 1994.
10. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. 36th Annual Symp. on Foundations of Computer Science*, pages 374–382, 1995.

On-Line Variable Sized Covering

Leah Epstein

School of Computer and Media Sciences, The Interdisciplinary Center, P.O.B. 167, 46150 Herzliya, Israel. lea@idc.ac.il

Abstract. We consider one-dimensional and multi-dimensional vector covering with variable sized bins.
In the one-dimensional case, we consider variable sized bin covering with bounded item sizes. For every finite set of bins B, and upper bound $1/m$ on the size of items for some integer m, we define a ratio $r(B,m)$. We prove this is the best possible competitive ratio for the set of bins B and the parameter m, by giving both an algorithm with competitive ratio $r(B,m)$, and an upper bound of $r(B,m)$ on the competitive ratio of any on-line deterministic or randomized algorithm. The ratio satisfies $r(B,m) \geq m/(m+1)$, and equals this number if all bins are of size 1.
For multi-dimensional vector covering we consider the case where each bin is a binary d-dimensional vector. It is known that if B contains a single bin which is all 1, then the best competitive ratio is $\Theta(1/d)$. We show an upper bound of $1/2^{d(1-o(1))}$ for the general problem, and consider four special case variants. We show an algorithm with optimal competitive ratio 1/2 for the model where each bin in B is a unit vector. We consider the model where B consists of all unit prefix vectors. Each bin has i leftmost components of 1, and all other components are 0. We show that this model is harder than the case of unit vector bins, by giving an upper bound of $O(1/\log d)$ on the competitive ratio of any deterministic or randomized algorithm. Next, we discuss the model where B contains all binary vectors. We show this model is easier than the model of one bin type which is all 1, by giving an algorithm of ratio $\Omega(1/\log d)$.
The most interesting multi-dimensional case is $d = 2$. Previous results give a 0.25-competitive algorithm for $B = \{(1,1)\}$, and an upper bound of 0.4 on the competitive ratio of any algorithm. In this paper we consider *all* other models for $d = 2$. For unit vectors, we give an optimal algorithm with competitive ratio 1/2. For unit prefix vectors we give an upper bound of 4/9 on the competitive ratio of any deterministic or randomized algorithm. For the model where B consists of all binary vectors, we design an algorithm with ratio larger than 0.4. These results show that all above relations between models hold for $d = 2$ as well.

1 Introduction

We consider on-line problems of covering variable sized bins. The one-dimensional version of this problem is defined as follows: We are given a finite set B of allowed bin sizes, each $b \in B$ satisfies $0 < b \leq 1$ and the largest element of B is 1.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 463–472, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Items with size in $(0, 1]$ arrive on-line, each item is to be assigned to a bin upon arrival. The algorithm may assign a new item to a previously used bin, or open a new bin of any size in B. A bin of size b is *covered* if the total size of items assigned to it is at least b. The goal of an algorithm is to maximize the sum of the sizes of all covered bins. This sum is the value of an algorithm. In this paper we consider bin covering with bounded item sizes. The size of each arriving item is at most $1/m$ where m is a fixed integer.

The multi-dimensional version is defined as follows: For a dimension d we are given a set of d-dimensional bins B, which is a subset of $\{0, 1\}^d$, i.e., all components of the allowed bins are binary. The "vector weight" of a d-dimensional vector b is the sum of its components, this applies both to multi-dimensional bins ("bin weights") and to multi-dimensional items ("item weights"). In the one-dimensional case the "bin weight" of a bin is simply its size and the "item weight" of an item is the size of the item. Vector items with size in $[0, 1]^d$ arrive on-line, and as in the one-dimensional case, each new item is to be assigned to a new or previously used bin upon arrival. A bin of size b is covered if the vector x, which is the sum of all items assigned to the bin, satisfies $x \geq b$. The goal of an algorithm is to maximize the sum of the bin weights of all covered bins. This number is the value of the algorithm. We study the general problem of arbitrary sets B, and also consider three interesting cases of multi-dimensional variable sized vector covering. The first case is where B consists of all possible binary vectors. The second case is when all bins are unit vectors (have only one non-zero coordinate). In the last case we consider a unit prefix sequence of bins, each bin has i leftmost 1 coordinates, and all others are 0.

Applications: Variable sized bin and vector covering relate to a case where a large number of jobs of a finite type are to be done (these are the bins). Each job is done by one worker or by a combined work of more than one worker (the workers are the items). An item represents the amounts of work a worker can do in a day. Each worker is assigned to one job (bin). The benefit of a job is the minimum amount of work to do it, which is proportional to its size. This scenario relates to one dimensional bin covering.

The scenario of vector covering is equivalent to the case where a job has d different qualities. We can normalize and assume that each quality which exists in the job has size 1, each type of job has to have a subset of the qualities, the different types of jobs are again represented by bins, which are d-dimensional binary vectors. The work of each worker is also represented by a d-dimensional vector which corresponds to the parts of each quality that the worker can do in one day.

Definition of competitive ratio: We measure the performance of an on-line algorithm by the competitive ratio. Denote the value of the on-line algorithm by V_{on}, and the value of an optimal off-line algorithm, that knows the sequence in advance, by V_{opt}. The competitive ratio is the supremum r ($r \leq 1$) for which $V_{on} \geq r \cdot V_{opt} - \eta$ is satisfied for every sequence (η is an additive constant which does not depend on the input). For randomized algorithms we replace V_{on} by $E(V_{on})$ and in this case the competitive ratio is the supremum r for which

$E(V_{on}) \geq rV_{opt} - \eta$ is satisfied for every sequence. An on-line algorithm is called optimal, if it has a competitive ratio which is the best possible competitive ratio. Note that in variable sized covering, the competitive ratio can be constant, or a function of B.

Known results: The *one-dimensional* version of the vector covering problem (for the case that all bins are of size 1) was first investigated in the thesis [2] of Assmann and in the journal article by Assmann, Johnson, Kleitman and Leung [3]. There it is proved that the greedy algorithm (that simply keeps putting items into the same bin until it is covered and then moves on to the next bin) has a worst case guarantee of 1/2. Moreover, two more sophisticated algorithms were derived with worst case ratios $\frac{2}{3}$ and $\frac{3}{4}$, respectively. Both of these sophisticated algorithms are based on presorting the items, and consequently are off-line algorithms. The greedy algorithm, however, is an on-line algorithm. Csirik and Totik [8] proved that in fact the greedy algorithm is a *best possible* on-line algorithm, since no on-line algorithm can have a worst case ratio that is strictly better than 1/2. Csirik, Frenk, Galambos and Rinnooy Kan [7] gave a probabilistic analysis of the one-dimensional bin covering and of the two-dimensional vector covering problem (for the case that all bins are of size $(1, 1)$). Gaizer [10] constructed an off-line approximation algorithm with worst case guarantee 1/2 for dimension $d = 2$. The article by Csirik and Frenk [6] summarizes all results on vector covering problems that were derived till 1990. The multi-dimensional vector covering problem was studied by Alon, Azar, Csirik, Epstein, Sevastianov, Vestjens and Woeginger in [1]. The paper considers the case where the only allowed bin is the "all 1" vector. This paper considers both on-line and off-line vector covering. [1] gives an on-line algorithm with competitive ratio $1/(2d)$ and an upper bound of $1/(d + 0.5)$ on the ratio of any deterministic or randomized algorithm. Finally Woeginger and Zhang [11] studied variable sized one-dimensional vector covering. They define a ratio $r(B)$ which is the minimum between the minimum ratio of two consecutive bin sizes (ignoring all bins smaller than 1/2) and the inverse of twice the size of the minimum sized bin among the bins which are larger or equal to 1/2. Clearly, this number is smaller than 1/2. They show this ratio $r(B)$ is optimal for a finite set of bins B. A survey on on-line packing and covering is given in [9].

Our results: We give some definitions in order to define the number $r(B, m)$. Let $B = \{b_1, \ldots, b_s\}$ be the set of allowed bin sizes, where $1 = b_1 > b_2 > \ldots > b_s$, and let m be an integer $m \geq 1$ such that $1/m$ is an upper bound on item sizes. For each $1 \leq i \leq s$, let $b_{i,j} = b_i/j$ and let $B_i(m)$ be the set of fractions of b_i between sizes $1/(2m)$ and $1/m$, that is $B_i(m) = \{b_{i,j} | 1 \leq j \leq 2m\} \cap [1/(2m), 1/m]$. We define a new set of bins by $C(m) = \cup_{1 \leq i \leq s} B_i(m)$. Enumerate the sizes of numbers in $C(m)$, $C(m) = \{c_1, \ldots, c_k\}$ where $1/m = c_1 > c_2 > \ldots > c_k = 1/(2m)$. For every element in $C(m)$, recall an original bin size that caused it to be inserted into $C(m)$. For c_i, let $b(c_i)$ be the smallest b_j in a way that there exists an integer y that satisfies $yc_i = b_j$. Now define $q(B, m) = \max\{c_i/c_{i+1} | 1 \leq i \leq k - 1\}$. Note that $1 + 1/m \geq q(B, m) > 1$. Finally define $r(B, m) = 1/q(B, m)$.

We show the following results for bin covering.

- For every finite set of bins B and integer $m \geq 1$, we give a deterministic algorithm with competitive ratio $r(B, m)$.
- For every finite set of bins B and integer $m \geq 1$, we give an upper bound of $r(B, m)$ on the competitive ratio of any algorithm.

These results show that $r(B, m)$ is the best possible competitive ratio for the set of bins B, and the upper bound $1/m$. This result reduces to the result of Woeginger and Zhang [11] for the case $m = 1$, and to the results of [3,8], if both $m = 1$ and $B = \{1\}$ hold. If only $B = \{1\}$ holds, then it follows from our result that the best competitive ratio is $m/(m + 1)$.

We show the following results for vector covering.

- An upper bound of $1/2^{d(1-o(1))}$ on the competitive ratio for the vector covering problem for arbitrary sets B.
- An algorithm with competitive ratio $1/2$ for the case where each bin in B is a unit vector.
- An upper bound of $1/2$ on the competitive ratio for this case
- For the case of unit prefix vectors, (i.e. bins are of the form $(1, \ldots, 1, 0, \ldots, 0)$ with i ones and $d-i$ zeros), and $|B| = d$) we show that this model is harder than the case where B consists of all unit vector bins (for large enough values of d) by giving an upper bound of $O(1/\log d)$ on the competitive ratio of any algorithm.
- For the same case we show that the relation between models holds also for $d = 2$ by giving an upper bound of $4/9$ on the competitive ratio of any algorithm for $d = 2$.
- For the case where B is the set of all possible binary vectors, we show that this model is easier than the basic model of one bin type which is all 1 (for large enough values of d) by giving an algorithm of ratio $\Omega(1/\log d)$.
- For the same case we show that the relation between models holds also for $d = 2$ by giving an algorithm of ratio larger than 0.4.

All upper bounds (negative results) in this paper hold both for deterministic and for randomized algorithms. In order to prove upper bounds we use an adaptation of Yao's theorem [12] which states that an upper bound of a deterministic algorithm against a fixed distribution on the input is also an upper bound for randomized algorithms. For maximization problems the bound is given by $E\left(V_{on}/V_{opt}\right)$ or by $E(V_{on})/E(V_{opt})$ [4,5]. All upper bounds are proved for algorithms that are allowed to have additive constants. In order to prove upper bounds for such algorithms, we show that all upper bounds hold for arbitrarily long sequences.

2 One-Dimensional Covering

In this section we show that for every set of one-dimensional bins B and integer $m \geq 1$, the ratio $r(B, m)$ is the best possible competitive ratio.

Theorem 1. *For every collection of bin sizes B, and parameter m, there exists an algorithm for bin covering with asymptotic worst case ratio $r(B, m)$. This result is best possible among all on-line randomized algorithms.*

We start with the algorithm. Later we prove the upper bound which shows its optimality.

For a given set of bins B, and parameter m, we will use q for $q(B,m)$, and r for $r(B,m)$. Let $t = \lceil -\log_2 m(q-1) \rceil$. Define a partition of the interval of items $(0, 1/m]$ into the following subintervals:

- For $1 \leq j \leq k-1$ and $0 \leq l \leq t-1$, let $I_{j,l} = (c_{j+1}/2^l, c_j/2^l]$.
- Let $I_{k,t} = (0, c_k/2^{t-1}]$.

We assign each interval $I_{j,l}$ a corresponding bin size $d_{j,l}$. We define $d_{k,t}$ to be $b_1 = 1$. For each other interval $I_{j,l}$ we define $d_{j,k} = b(c_{j+1})$. The algorithm keeps one open bin of for each interval, this bin is used only for packing items in the interval. Each new item is classified and assigned to the corresponding bin. When a bin is covered, it is closed, and a new bin of the same size is opened and used for this interval.

Lemma 1. *The competitive ratio of the above algorithm is at least r*

Proof. We show that each covered bin is covered by at most q times its size. For every first type interval $I_{j,l}$ we show that its corresponding closed bins are covered by at most c_j/c_{j+1} times their size. Let $b(c_{j+1}) = yc_{j+1}$, then each such bin is covered by exactly $y2^l$ items of size at most $c_j/2^l$, and the total size of the items that cover it is at most yc_j. Thus the ratio between the size of items that cover the bin and the size of the bin is at most c_j/c_{j+1} as required. Since $q \geq c_j/c_{j+1}$, the bin is covered by at most q times its size.

For the interval $I_{k,t}$, since all items are bounded by $1/(2^t m)$, a covered bin is covered by items with total size at most $1 + 1/(2^t m)$. Since $2^t \geq 1/(m(q-1))$, $1 + 1/(2^t m) \leq q - 1 + 1 = q$, and each such bin is covered by at most q, which is q times a size of a unit bin.

The total number of bins that are never closed is $O(kt)$, which is $O(mst)$ where $s = |B|$, this number depends solely on B and m, and is an additive constant.

Lemma 2. *The competitive ratio of any deterministic or randomized algorithm for bin covering is at most r.*

Proof. We start with the deterministic case. Let j be an integer such that $c_j/c_{j+1} = q$. Let y be an integer such that $yc_j = b(c_j)$. Let N be a large integer.

In order to define a list of items, we define a set of sizes D. Let $D = B \cup C'$ where $C' = \{xc' | x \in \{1, \ldots, 2m\}, c' \in C\} \cap [1/2m, 1]$. Enumerate the numbers in D, $D = \{d_1, \ldots, d_p\}$ where $d_1 > d_2 > \ldots > d_p$. Now define $\varepsilon > 0$ to be a number that satisfies $2m \cdot 2^N \varepsilon < b_s$ and for every $1 \leq j \leq p-1$, $2m \cdot 2^N \varepsilon < d_j - d_{j+1}$. The list contains $2^N y$ items with size ε (sand) followed by $2^{N-i} y$ items with size $c_j - 2^i \varepsilon$ for a fixed i, $0 \leq i \leq N$.

The optimal off-line assignment uses bins with size $b(c_j)$ to pack all items. Each bin contains y items with size $c_j - 2^i \varepsilon$ and $2^i y$ items with size ε. This covers 2^{N-i} bins and $V_{opt} = 2^{N-i} b(c_j) = 2^{N-i} y c_j$. We change the packing of the on-line algorithm without reducing the value of the on-line algorithm in such a way

that the on-line algorithm will use only bins of weights c_j and c_{j+1}. Consider a bin of the on-line algorithm of size b which is covered, i.e. it contributes some amount to the on-line algorithm value. Since $y \leq 2m$, and $b \geq b_s > 2m2^N\varepsilon$, the bin contains at least one big item. Let $a > 0$ be the number of big items in the bin. Assume that $a \leq 2m$, otherwise remove items and let $a = 2m$, since items are larger than $1/2m$ and the bin is of size at most 1, then the bin is still covered.

Let μ be the total size of small items in the bin. If $\mu \geq 2^i a\varepsilon$ replace the bin by a bins of size c_j containing each one big item, and $\lfloor \mu/(a\varepsilon) \rfloor$ small items. Otherwise, partition the items in the same way, but use bins of weight c_{j+1} instead.

We show that the new bins are covered and that the value of the on-line algorithm is not reduced, i.e. $b \leq ac_j$ in the first case and $b \leq ac_{j+1}$ in the second case. In the first case since $\mu/a \geq 2^i\varepsilon$, the total weight of items in a new bin is $(c_j - 2^i\varepsilon) + \lfloor \mu/(a\varepsilon) \rfloor \varepsilon \geq c_j$. Hence each new bin is covered. On the other hand $b \leq ac_j + y2^N\varepsilon$. According to the definition of D, $ac_j \in D$. Assume to the contrary that $b > ac_j$. Since $b \in B$ then $b \in D$, thus $b - ac_j > y2^N\varepsilon$ and hence the bin would not be covered. We conclude that $b \leq ac_j$, and the replacement of b by a bins of size c_j does not change the on-line algorithm value and each new bin of size c_j is covered by at least c_j. In the second case, If $\mu < 2^i a\varepsilon$, the weight of an item in a new bin is at least $c_j - 2^i\varepsilon > c_{j+1}$. Hence each new bin is covered. On the other hand $b < ac_j$. Assume to the contrary that $ac_{j+1} < b < ac_j$, then $b/a \in C$, contradicting the fact that c_j in the successor of c_{j+1} in C. We conclude that new bins are covered, the on-line algorithm value was not reduced, and now the on-line algorithm uses only bins of sizes c_j and c_{j+1}.

For $0 \leq i < N$, let Z_i be the number of on-line algorithm bins with the amount of least $2^i\varepsilon$ weight of sand but less than $2^{i+1}\varepsilon$. Let Z_N be the number of on-line algorithm bins with weight of sand at least 2^N. We have $\sum_{i=0}^{N} Z_i 2^i \leq y2^N$. The on-line algorithm value would be $c_j(Z_i+Z_{i+1}+\ldots+Z_N)+c_{j+1}(2^{N-i}y-(Z_i + Z_{i+1} + \ldots + Z_N)) = yc_{j+1}2^{N-i} + (c_j - c_{j+1})(Z_i + Z_{i+1} + \ldots + Z_n)$.

We multiply the on-line and off-line algorithms values for every $i \geq 1$ by $2^{i-1}/2^N$, and the value for $i = 0$ by $1/2^N$. For the on-line algorithm we get $yc_{j+1}(\sum_{i=1}^{N} 2^{N-i}2^{i-1}/2^N + 2^N/2^N) + (c_j - c_{j+1})\sum_{i=0}^{N} Z_i 2^i/2^N \leq y(c_{j+1}(N + 2)/2 + (c_j - c_{j+1})) = y/2((N + 2)c_{j+1} + 2c_j - 2c_{j+1}) = y/2(Nc_{j+1} + 2c_j)$
. For the optimal off-line algorithm we get $\sum_{i=1}^{N} 2^{i-1}/2^N \cdot 2^{N-i}yc_j + 1/2 \cdot 2^N yc_j = (yc_j/2^N)(N + 2)2^{N-1} = yc_j(N + 2)/2$.

If the competitive ratio is r and $q = 1/r$, then also for the convex sums $yc_j(N + 2)/2 \leq qy/2(Nc_{j+1} + 2c_j)$ or $q \geq (N + 2)/((N(c_{j+1}/c_j) + 2)$. Hence $r = c_{j+1}/c_j + \delta_N$, where δ_N is arbitrarily close to 0, for large enough values of N.

For the randomized case we use the adaptation to Yao's theorem, and the probabilities are the factors we multiplied by in the deterministic case. We get that $E(V_{on})/E(V_{opt}) \leq c_{j+1}/c_j$.

3 Multi-dimensional Vector Covering

In this section we consider variable sized vector covering. In the classical vector covering problem, B contains a single bin which is all 1. We study a simple extension of this model, and allow each bin to be a binary d-dimensional vector. We start by showing upper bounds on the competitive ratio for arbitrary sets B. The proof of the following Lemma is omitted.

Lemma 3. *The competitive ratio of any deterministic or randomized on-line algorithm for binary variable sized vector covering is at most* $1/2^{d(1-o(1))}$.

Due to this negative result, we do not study the general model, but restrict ourselves to specific interesting cases of B. We show some relations between the different models.

3.1 Covering a Single Type of Bin

We start with an easy model where B consists of a single type of bin b. Assume that the number of non-zero components in b is i, then we can reduce this problem to the case of vector covering bins of dimension i, where all components are 1 (the basic vector covering problem). This is true since both the on-line and off-line algorithms do not use the coordinates which are zero in the vector b. Thus the best competitive ratio for this case is $\Theta(1/i)$ [1].

3.2 Covering Unit Vector Bins

In this model we let B consist of bins which have one non-zero component. We show that for any such set of bins, the best competitive ratio is 1/2. We consider this model since this is a case where all bins have the same bin weight.

The upper bound (negative result) follows from the bound of [8] by picking one $b \in B$, where the ith coordinate of b is non-zero, and giving a sequence where all vectors are d-dimensional, with all coordinates zero, except for the ith coordinate which is the same as in [8]. We can also give an easier proof for the case where $|B| > 1$, we omit this proof due to space restrictions.

Lemma 4. *If B consists of unit vector bins only, and $|B| > 1$, then the competitive ratio of any deterministic or randomized on-line algorithm is at most 1/2.*

We give an optimal greedy algorithm. We assume that $|B| = d$ and it contains all bins of unit bin weight. Otherwise we just cancel the coordinates that do not appear in any bin (are zero in all bins), since no algorithm can benefit from these coordinates, and treat the input as a lower dimension vector.

The algorithm has at most d open bins at a time. Each arriving vector is classified to a type among $\{1, 2, \ldots, d\}$ according to its largest component.

Denote a bin whose ith coordinate is 1 by b_i. After an arriving item is classified to a type j, then if there is an open bin of size b_j, the item is assigned to

it, and if after the item is assigned to a bin, the bin is covered, we close the bin (and never use it again).

If no bin of size b_j is open, open a new one and continue in the same way.

Theorem 2. *The competitive ratio of the greedy algorithm is at least* $\frac{1}{2}$.

Proof. For the optimal off-line algorithm, define by S_{opt}, the active area of items, that is, the area of items that was used to cover bins (i.e. the area that is not wasted, which is in the case of unit vectors, 1 for each covered bin). Clearly, $V_{opt} = S_{opt}$. Since all bins have only one non-zero coordinate, the active area of each item is in one coordinate. For an item a, let $a_{\max} = \max_{1 \le i \le d} a_i$. Clearly, $S_{opt} \le \sum_{a \in A} a_{\max}$, where A is the sequence of items.

On the other hand, consider the area $\sum_{a \in A} a_{\max}$, the on-line algorithm packed according to the maximum component.

Let α_i be the number of bins of type b_i that the on-line algorithm managed to cover. For each such bin, the ith coordinate is covered by less than 2 (since at the time the last item was assigned there, the coordinate was less than 1, and the new item adds at most 1). There is at most one open bin which is never closed, and its ith coordinate is also less than 1. Hence $\sum_{a \in A_i} a_{\max} \le 2\alpha_i + 1$, where A_i are items that were classified to type i. Summing over all $1 \le i \le d$ we get: $\sum_{a \in A} a_{\max} \le 2V_{on} + d$, which gives $V_{on} \ge \frac{1}{2} V_{opt} - \frac{d}{2}$.

3.3 Covering Unit Prefix Bins

We define a model which is similar to the previous one. Let B_1 be a set that contains exactly d possible bin sizes. Bin i, which also has bin weight i, is a d-dimensional vector whose i leftmost coordinates are 1, and all others are 0. We show that using this set of allowed bins instead of d unit vector bins, makes the problem of on-line covering harder.

Theorem 3. *The competitive ratio of any deterministic or randomized on-line algorithm for covering the increasing set of bins has competitive ratio of at most* $O(1/\log d)$.

Proof. The upper bound sequence consists of $2k$ items with size $a = (1 - \varepsilon, 0, 0, \ldots, 0)$ and then with probability p_i (which we fix later), $2k$ items with size $(\varepsilon, 1, \ldots, 1, 0, \ldots, 0)$ arrive ($d - i$ zeros), and for $2 \le i \le d$, with probability q_i, (which is also fixed later), k items with size $(0, 1, \ldots, 1, 0, \ldots, 0)$ arrive ($d - i$ zeros). q_1 is the probability that no further items (except the first $2k$ items) arrive. Hence the probabilities should be fixed so that $\sum_{i=1}^{d} (p_i + q_i) = 1$.

Let α_i be the number of bins of bin weight i that the on-line algorithm opens for one item of size a, and β_i to be the number of bins of bin weight i it opens, assigning two items with size a to each.

Let us calculate the competitive ratio for each case: Consider the cases where items with $d - i$ rightmost zero coordinates arrive. With probability p_i, the on-line algorithm manages to cover all bins of weight of at most i that it opened.

The optimal off-line assignment would use only bins of bin weight i, and would assign one item of each type to every bin. Thus $C_{opt} = 2ki$, and $C_{on} =$

$\sum_{j=1}^{i}(\alpha_j+\beta_j)\cdot j$. On the other hand, with probability q_i, the on-line algorithm would cover only bins with at least two items with size a (otherwise the leftmost component is not covered).

The optimal off-line assignment would use only bins of bin weight i, and would assign two items with size a, and one item of the other type to each bin. Thus $C_{opt} = ki$ and $C_{on} = \sum_{j=1}^{i}\beta_j \cdot j$. According to the initial on-line assignment, we have $\sum_{i=1}^{d}(\alpha_i + 2\beta_i) \leq 2k$. Let $p_i = q_i = \frac{1}{2i(i+1)}$ for $i < d$ and $p_d = q_d = \frac{1}{2d}$.

The expectation of the optimal off-line value is $E(V_{opt}) = \sum_{i=1}^{d}(p_i 2ki + q_i ki) = 3k\Big(\sum_{i=1}^{d-1}\frac{1}{2(i+1)} + \frac{1}{2}\Big) == \frac{3}{2}k\Big(\sum_{i=1}^{d-1}\frac{1}{i+1}+1\Big) = \frac{3}{2}k\Big(\sum_{i=1}^{d}\frac{1}{i}\Big) \geq \frac{3}{2}k\ln d$. It is possible to show that the expectation of the on-line value satisfies $E(V_{on}) \leq k$. The probabilities are valid since $\sum_{i=1}^{d}(p_i+q_i) = \sum_{i=1}^{d-1}\frac{1}{i(i+1)}+\frac{1}{d} = \sum_{i=1}^{d-1}\Big(\frac{1}{i} - \frac{1}{i+1}\Big) + \frac{1}{d} = 1$. Thus the competitive ratio is at most $\frac{k}{\frac{3}{2}k\ln d} = \frac{2k}{3k\ln d} = O\Big(\frac{1}{\log d}\Big)$.

We show that this problem is significantly harder than unit vector bins, not only asymptotically but for $d = 2$ as well (in this case $B = \{(1,1),(1,0)\}$). The Lemma follows from a more careful examination of the general sequence for $d = 2$. The full proof is omitted.

Lemma 5. *The competitive ratio of any deterministic or randomized algorithm for $d = 2$ and unit prefix bins is at most $4/9$.*

3.4 Covering All Possible Bins

In this model, B consists of $2^d - 1$ bins, i.e. each binary d-dimensional bin may be used. We show that the basic covering model (the only allowed bin is the "all 1" bin) is harder than this model. We define the following algorithm.

Let $k = \lceil \log_2(d^2) \rceil$. We define type vectors by vectors in the set A^d where $A = \{0\} \cup \{\frac{1}{2^i} \mid 0 \leq i \leq k\}$. A^d consists of $(k+2)^d$ different type vectors. The algorithm keeps one open bin of a corresponding size for each vector type. To determine the bin vector for a certain type vector we do the following:
For each $0 \leq s \leq k$, let α_s be the sum of sizes of all components of size $\frac{1}{2^s}$ of the type vector, which is $\frac{m}{2^s}$, where m is the number of $\frac{1}{2^s}$ components. Let s_1 be an integer for which α_s is maximized. The corresponding bin would have zeros in all coordinates which are less than $\frac{1}{2^{s_1}}$ in the type vector, and 1 in all coordinates which are at least $\frac{1}{2^{s_1}}$.

We keep one open corresponding bin for each possible type vector. On arrival of an item vector a, a is identified with a type vector, then it is assigned to a bin that corresponds to this type. A covered bin is closed and replaced by a new bin of the same size.

For each arriving item vector a, compute its type vector in the following way: Let i be the coordinate such that $a_i = \max_{1\leq j\leq d} a_j$. Let a' be the vector $\frac{a}{a_i}$. Replace each coordinate a'_j by the maximum number $\frac{1}{2^s}$, such that $a'_j \geq \frac{1}{2^s}$. If $s > k$, then replace a'_j by zero. The new vector a'' is the type vector of a. Let

$a''' = a'' \cdot a_i$. a is assigned to the open bin which corresponds to the type vector a''.

Theorem 4. *The above algorithm has competitive ratio* $\Omega(\frac{1}{\log d})$.

The proof of the Theorem 4 is omitted. It is also possible to design an algorithm for $d = 2$, and prove the following theorem.

Theorem 5. *There exists an algorithm for* $d = 2$*, for covering with all possible bins, which has competitive ratio strictly above* 0.4*. The algorithm divides items into ranges, and packs each range separately.*

Acknowledgments

I would like to thank Gerhard Woeginger for introducing me to some of the problems discussed in this paper. I would also like to thank Yossi Azar for helpful discussions.

References

1. N. Alon, Y. Azar, J. Csirik, L. Epstein, S.V. Sevastianov, A.P.A. Vestjens, and G.J. Woeginger. On-line and off-line approximation algorithms for vector covering problems. *Algorithmica*, 21:104–118, 1998.
2. S.F. Assmann. Problems in discrete applied mathematics. Technical report, Doctoral Dissertation, Mathematics Department, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1983.
3. S.F. Assmann, D.S. Johnson, D.J. Kleitman, and J.Y.-T. Leung. On a dual version of the one-dimensional bin packing problem. *J. Algorithms*, 5:502–525, 1984.
4. A. Borodin and R. El-Yaniv. On randomization in online computations. In *Computational Complexity*, 1997.
5. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.
6. J. Csirik and J.B.G. Frenk. A dual version of bin packing. *Algorithms Review*, 1:87–95, 1990.
7. J. Csirik, J.B.G. Frenk, G. Galambos, and A.H.G. Rinnooy Kan. Probabilistic analysis of algorithms for dual bin packing problems. *J. Algorithms*, 12:189–203, 1991.
8. J. Csirik and V. Totik. On-line algorithms for a dual version of bin packing. *Discr. Appl. Math.*, 21:163–167, 1988.
9. J. Csirik and G.J. Woeginger. On-line packing and covering problems. In *A. Fiat and G. J. Woeginger, editors,* Online Algorithms: The State of the Art, volume 1442 of *LNCS*, pages 154–177. Springer-Verlag, 1998.
10. T. Gaizer. An algorithm for the 2d dual bin packing problem. Unpublished manuscript, University of Szeged, Hungary, 1989.
11. Gerhard J. Woeginger and Guochuan Zhang. Optimal on-line algorithms for variable-sized bin covering. *Operations Research Letters*, 25:47–50, 1999.
12. A.C.C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, pages 222–227, 1977.

On Testing for Zero Polynomials by a Set of Points with Bounded Precision

Jin-Yi Cai* and Eric Bach**

Computer Sciences Department
University of Wisconsin – Madison
1210 West Dayton Street
Madison, WI 53706
{jyc,bach}@cs.wisc.edu

Abstract. We consider a general methodology proposed by Chen and Kao [4] for testing polynomial identities. We prove that the test *cannot* be completely derandomized by any specified set of rational approximations to algebraic numbers up to a polynomial number of bits. The proof is a direct application of Dirichlet's box principle. We also give some number theoretic estimates for the likelihood of a multiplicatively independent sequence of integers which can be used in their algorithm.

1 Introduction

Randomization has been used very successfully in the field of algorithm design and complexity analysis. The first widely acclaimed use of randomization as a computational resource with provable polynomial time performance bounds can be traced to Solovay and Strassen [13] and to Rabin [11] following work by G. Miller [8]. These results were for testing primes. Another early work using randomness to achieve provable bounds in computation is the Schwartz-Zippel Theorem around 1978 [12] [14]:

Theorem 1 (Schwartz-Zippel). *Let $F(x_1, \ldots, x_n) \in \mathbf{F}[x_1, \ldots, x_n]$ be a multivariate polynomial of total degree d, where $\mathbf{F}$ is any fixed field. Suppose F is not identically zero. Then, for any fixed finite set $S \subseteq \mathbf{F}$,*

$$\Pr[F(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|},$$

where $r_1, \ldots, r_n$ are chosen independently and uniformly at random from S.

This elegant result is used to test for polynomial identities. Thus, if we wish to test for $f(\mathbf{x}) = g(\mathbf{x})$, we simply pick a set of random inputs $\mathbf{r}$ and check that $f(\mathbf{r}) = g(\mathbf{r})$, with the understanding that if $f(\mathbf{x}) = g(\mathbf{x})$, then certainly

* Research supported in part by grants from NSF CCR-9820806, NSF SBE-INT Japan Program and a J.S. Guggenheim Fellowship.
** Research supported in part by grants from NSF CCR-9988202.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 473–482, 2001.
© Springer-Verlag Berlin Heidelberg 2001

$f(\mathbf{r}) = g(\mathbf{r})$ for any $\mathbf{r}$, and, if $f(\mathbf{x}) \neq g(\mathbf{x})$, then $f(\mathbf{r}) \neq g(\mathbf{r})$ with high probability. The error probability is only on one side, and can be reduced by either choosing a larger set S, or, as a more general principle, by running the probabilistic test many times. There are more advanced techniques for amplification (see [9,15]).

The Schwartz-Zippel method is a general technique for testing polynomial identities. Not only it can be used in straightforward algebraic settings, but also it can be used to give elegant probabilistic algorithms for combinatorial problems. A well known application is the probabilistic NC^2 algorithm for the perfect matching problem by Lovász [7].

A *perfect matching* in a graph G is a set M of edges, such that every vertex of G is incident to exactly one edge in M. The perfect matching problem is to decide if a given graph has a perfect matching. Tutte gave a beautiful characterization of the existence of perfect matchings in a graph G. Define the *Tutte* matrix T as follows. Suppose G has n vertices and m edges. Let $\{x_{i,j}\}$ be a set of m indeterminants, one for each (undirected) edge (i, j). For any edge (i, j), with $i < j$, put an $x_{i,j}$ in the (i, j) position and $-x_{i,j}$ in the (j, i) position, and 0 elsewhere.

Theorem 2 (Tutte). *G has a perfect matching if and only if* $\det T \neq 0$.

Then Lovász's RNC^2 algorithm for the perfect matching problem is to apply the probabilistic test for polynomial identity and test for zero. More specifically, his algorithm assigns to each indeterminant a uniformly and independently chosen integer in the range $\{1, \ldots, 2n\}$. Note that the total degree of $\det T$ is (at most) n, and thus the algorithm achieves success probability $1/2$ with $m \cdot \lceil \log_2(2n) \rceil$ random bits.

There are other randomized fast parallel algorithms for the perfect matching problem. For instance, Chari, Rohatgi and Srinivasan [3] gave an alternative algorithm that uses fewer random bits, but the processor count is a large polynomial in n (it is estimated that the algorithm by Chari et al needs to compute determinant and inverse of $n \times n$ matrices of entries of size up to n^7; see [4].) We note that it has been a long standing open problem to achieve a deterministic NC^2 algorithm for the perfect matching problem.

Recently Chen and Kao [4] proposed a new general methodology for testing whether a polynomial with integer coefficients is identically zero. Their method is based on a theorem on the non-vanishing of a polynomial on algebraic numbers. Their motivation is to derandomize the RNC^2 algorithm for the perfect matching problem. The general idea of their approach is that, instead of evaluating the polynomial at random integer inputs, it can be evaluated at various algebraic numbers.

More specifically, let $p_1 = 2, p_2, \ldots, p_n$ be the smallest n primes, and let $f(x_1, \ldots, x_n)$ be a non-zero multilinear polynomial with integer coefficients. Then

$$f(\pm\sqrt{p_1}, \pm\sqrt{p_2}, \ldots, \pm\sqrt{p_n}) \neq 0.$$

More generally, let $f(x_1, \ldots, x_n)$ have degree d_i on x_i. Let $k_i = \lceil \log_2(d_i+1) \rceil$, and $s = \sum_{i=1}^{n} k_i$. Let $q_{1,1}, \ldots, q_{1,k_1}, \ldots, q_{n,1}, \ldots, q_{n,k_n}$ be any s square-free positive

integers which are pair-wise relatively prime, and let $b_{1,1}, \ldots, b_{1,k_1}, \ldots, b_{n,1}, \ldots, b_{n,k_n}$ be any s bits from $\{0,1\}$. Then f is non-zero if and only if

$$f\left(\sum_{j=1}^{k_1}(-1)^{b_{1,j}}\sqrt{q_{1,j}}, \sum_{j=1}^{k_2}(-1)^{b_{2,j}}\sqrt{q_{2,j}}, \ldots, \sum_{j=1}^{k_n}(-1)^{b_{n,j}}\sqrt{q_{n,j}}\right) \neq 0.$$

This does give a deterministic test, albeit one which involves irrational numbers, and thus its exact computation is computationally infeasible.

To overcome this difficulty, Chen and Kao used a certain random combination of rational approximations to the irrational algebraic numbers. Indeed, in the above deterministic test there is no need for the s bits $b_{i,j}$ at all; these were included for the purpose of using randomization. More specifically, they used a random choice of the s bits, with a truncation of the quadratic irrational numbers to a prespecified polynomial number of bits ℓ. They showed that to achieve good success probability, $\ell = \lceil \log_2(c + cd) \rceil + d\lceil \log_2 n \rceil + 2d\lceil \log_2 s \rceil + d + 1$ suffices, where d is the total degree of f and c is its maximum absolute value of coefficients. In particular, they suggest using simply the truncations of $(\pm\sqrt{p_1}, \pm\sqrt{p_2}, \ldots, \pm\sqrt{p_n})$ for multilinear polynomials, where the signs $\pm$ are randomly chosen.

With this method, they arrived at their randomized NC^2 algorithm for the perfect matching problem. It turns out that the number of random bits used is only $\sum_{i=1}^{n}\lceil \log_2 \max\{1, n_i\} \rceil$, where n_i is the number of neighbors v_j of the i-th vertex v_i in G such that $j > i$. This allowed them to achieve an RNC^2 algorithm that uses fewer random bits without doing more work than all previous randomized NC algorithms for the perfect matching problem. They also used their general methodology for several other problems.

In this paper we show that the use of prime square roots $\sqrt{p_1}, \sqrt{p_2}, \ldots, \sqrt{p_n}$ is unnecessarily restrictive. Even if we use square-free positive integers which are pair-wise relatively prime, it is still unnecessarily restrictive. Instead we only need numbers that are multiplicatively independent modulo squares. Note that it is computationally easy to test for this property of multiplicative independence modulo squares [1]. Moreover, for polynomially many random integers, it is exponentially unlikely that they are pair-wise relatively prime. On the other hand, it is superpolynomially unlikely that they are multiplicatively dependent modulo squares. We prove these facts in Section 3.

While their algorithm reduces the number of random bits required, the possibility remains that perhaps one might be able to eliminate randomness altogether, by choosing carefully a specific set of signs in the rational combination of irrational numbers and a sufficiently good rational approximation to each of the irrational values, and then evaluating. As their methodology is a general one, the general question then is whether there are such universal choices. The authors of [4] consider this the most intriguing problem left open from their approach [5].

In Section 4 we show that this is in general impossible.

2 Algebraic Preliminaries

Let $\mathbf{Q}$ be the field of rational numbers. Let $\mathbf{Q}^*$ be the multiplicative group of $\mathbf{Q}$, and $\mathbf{Q}^{*2}$ be the subgroup of squares, i.e., $\mathbf{Q}^{*2}$ is the image of $x \mapsto x^2$ from $\mathbf{Q}^*$. By the unique factorization of integers, the quotient group $R = \mathbf{Q}^*/\mathbf{Q}^{*2}$ is a direct sum of countably many copies of $\mathbf{Z}_2$ generated by $p_0 = -1$, and by $p_1 = 2, \ldots, p_n, \ldots$, respectively, where p_n is the n-th prime, each of which has order 2 in R.

Let $\beta_1, \beta_2, \ldots, \beta_m$ be any m *multiplicatively independent* elements in R, i.e., for any non-empty $S \subseteq \{1, \ldots, m\}$, $\prod_{i \in S} \beta_i \neq 1$ in R. Results of the following type are generally referred to as Kummer theory (see Lang [6] Chapter VIII Section 8). However we present the following elementary proof.

Lemma 1. *Let* $\mathbf{Q}_m = \mathbf{Q}(\sqrt{\beta_1}, \sqrt{\beta_2}, \ldots, \sqrt{\beta_m})$. *Then* $[\mathbf{Q}_m : \mathbf{Q}] = 2^m$.

Proof. Write each β_j as a 0-1 vector of the exponents mod 2 over primes as follows. By unique factorization, $\beta_j = \prod_{i \geq 0} p_i^{e_i} r^2$, where $r^2 \in \mathbf{Q}^{*2}$ and e_i mod 2 are uniquely defined. (Here $p_0 = -1$ and $p_i, i \geq 1$ are all the primes.) The uniqueness claim is just the elementary statement that for distinct primes p_i, $\pm \prod p_i \neq 1$ is not a square in $\mathbf{Q}^*$. Then the 0-1 vector for β_j is $(e_0, e_1, \ldots)$. It is actually a finite vector, one entry for every prime appearing in β_j; or one can think of it as an infinite 0-1 vector, with all but finitely many entires 0. This establishes an isomorphism between R and $\oplus \mathbf{Z}_2$. Now one can perform Gaussian elimination on the matrix over $\mathbf{Z}_2$ whose rows are these 0-1 vectors for $\beta_1, \beta_2, \ldots, \beta_m$. (In matrix terms, the elementary row/column operations are: exchange of two rows or columns, adding mod 2 of one row to another.) Being *multiplicatively independent* is equivalent to the rank of this matrix being m over $\mathbf{Z}_2$. We can arrive at a normal form after Gaussian elimination $[I_m \ B]$, where I_m is the $m \times m$ identity matrix and B is some matrix over $\mathbf{Z}_2$ with m rows. If $\alpha_1, \alpha_2, \ldots, \alpha_m$ are the elements corresponding to the rows, then there are m distinct $q_1, q_2, \ldots, q_m \in \{p_0, p_1, \ldots\}$, such that $\alpha_j = q_j r_j$, where every r_j is a product of p_i's other than $q_1, q_2, \ldots, q_m$. In terms of field extension:

$$\mathbf{Q}_m = \mathbf{Q}(\sqrt{\alpha_1}, \sqrt{\alpha_2}, \ldots, \sqrt{\alpha_m}).$$

Denote $\mathbf{Q}_j = \mathbf{Q}(\sqrt{\alpha_1}, \sqrt{\alpha_2}, \ldots, \sqrt{\alpha_j})$.

Clearly $[\mathbf{Q}_m : \mathbf{Q}] \leq 2^m$ as $[\mathbf{Q}_j : \mathbf{Q}_{j-1}] \leq [\mathbf{Q}(\sqrt{\alpha_j}) : \mathbf{Q}] = 2$. There is an injective homomorphism from the Galois group $G = \mathrm{Gal}(\mathbf{Q}_m/\mathbf{Q})$ to the permutation group which maps $\sqrt{\alpha_j}$ to $\pm\sqrt{\alpha_j}$, for every j. The full permutation group is the dual group of $\mathbf{Z}_2^m$ which has order 2^m. We want to show that this homomorphism is surjective, and thus G is isomorphic to the dual of $\mathbf{Z}_2^m$ and in particular has order 2^m as well. Being the splitting field of $\prod_{j=1}^m (X^2 - \alpha_j)$, $\mathbf{Q}_m$ is *Galois* over $\mathbf{Q}$, and thus by the fundamental Galois correspondence $[\mathbf{Q}_m : \mathbf{Q}] = |G| = 2^m$.

To show that $|G| = 2^m$, we represent each $\sigma \in G$ as a 0-1 vector $(\sigma_j)_{j=1}^m$ by the action on $\{\sqrt{\alpha_j}\}$, $\sigma(\sqrt{\alpha_j}) = (-1)^{\sigma_j}\sqrt{\alpha_j}$. Since each α_j has a distinct q_j, we have the following important property of G: For any non-empty subset

of indices $I \subseteq \{1, \ldots, m\}$, let $\alpha_I = \prod_{j \in I} \alpha_j$, then there is an automorphism $\tau \in \mathrm{Gal}(\mathbf{Q}(\sqrt{\alpha_I})/\mathbf{Q})$ which maps $\sqrt{\alpha_I}$ to $-\sqrt{\alpha_I}$. This τ extends to some σ in G. In terms of the 0-1 vectors (σ_j), for every non-empty subset I of $\{1, \ldots, m\}$, there is some (σ_j), such that $\sum_{j \in I} \sigma_j = 1 \bmod 2$.

We claim that the 0-1 vectors obtained from G include m unit vectors where the i-th unit vector has its i-th entry 1 and 0 elsewhere. Inductively, we assume there are $v^{(i)}$, $1 \le i \le j$, whose restriction to the first j coordinates are the first j unit vectors. The base case is established by taking $I = \{1\}$. For $j+1$, take $I = \{i \mid 1 \le i \le j,\ v^{(i)}_{j+1} = 1\} \cup \{j+1\} \subseteq \{1, 2, \ldots, j+1\}$. Then there is a u from G, such that $\sum_{k \in I} u_k = 1 \bmod 2$. Define

$$v^{(j+1)} = u + \sum_{i : 1 \le i \le j, u_i = 1} v^{(i)},$$

then $v^{(j+1)}$ satisfies our requirement.

This is fairly clear. Formally, starting with $k \le j$,

$$v^{(j+1)}_k = u_k + \sum_{i : 1 \le i \le j, u_i = 1} v^{(i)}_k = u_k + \sum_i 1_{[u_i = 1, i = k]} = 0 \bmod 2.$$

For coordinate $j+1$,

$$\begin{aligned} v^{(j+1)}_{j+1} &= \sum_{k \in I} v^{(j+1)}_k \qquad \text{(adding zero terms)} \\ &= \sum_{k : 1 \le k \le j, k \in I, u_k = 1} \left(u_k + v^{(k)}_k + v^{(k)}_{j+1} \right) + u_{j+1} \\ &= \sum_{k \in I} u_k \\ &= 1 \bmod 2. \end{aligned}$$

To complete the induction, we replace $v^{(i)}$ by $v^{(i)} + v^{(j+1)}$ for $i \in I, i \le j$. □

We actually established more, namely we computed the Galois group $G = \mathrm{Gal}(\mathbf{Q}_m/\mathbf{Q})$. Moreover, reverting back to the multiplicatively independent set $\{\beta_1, \ldots, \beta_n\}$, for any j, the proof gives a permutation $\sigma \in G$ which flips $\sqrt{\beta_j}$ to $-\sqrt{\beta_j}$ but fixes all others. These generate G as a free abelian group of exponent 2. Hence we have

Corollary 1. *For any sequence $a_1, a_2, \ldots, a_m \in \mathbf{Q}^*$, $\mathbf{Q}(\sum_{i=1}^m a_i \sqrt{\beta_i}) = \mathbf{Q}_m$. In particular this is true for any sign sequences $a_i = \pm 1$.*

Proof. If $\mathbf{Q}(\sum_{i=1}^m a_i \sqrt{\beta_i})$ were a proper subfield of $\mathbf{Q}_m$, there would be a non-trivial automorphism of G which fixes $\sum_{i=1}^m a_i \sqrt{\beta_i}$. But a non-trivial automorphism must flip *some* $\sqrt{\beta_i}$, leading to a non-trivial linear relation $\sum_S 2a_i \sqrt{\beta_i} = 0$, where $S \neq \emptyset$. Knowing the structure of G, this is obviously impossible as we can apply *some* σ which flips one $\sqrt{\beta_{i_0}}$ where $i_0 \in S$ and fixes all others, and get $\sqrt{\beta_{i_0}} = 0$. □

Let $f(x_1, \ldots, x_n) \in \mathbf{Q}[x_1, \ldots, x_n]$ have degree d_i on x_i. Let $k_i = \lceil \log_2(d_i + 1) \rceil$, and $s = \sum_{i=1}^{n} k_i$. Let $q_{1,1}, \ldots, q_{1,k_1}, \ldots, q_{n,1}, \ldots, q_{n,k_n}$ be any s multiplicatively independent elements of $R = \mathbf{Q}^*/\mathbf{Q}^{*2}$, and $a_{1,1}, \ldots, a_{1,k_1}, \ldots, a_{n,1}, \ldots, a_{n,k_n}$ be any s elements of $\mathbf{Q}^*$. Denote by $q_i = \sum_{j=1}^{k_i} a_{i,j}\sqrt{q_{i,j}}$, then

Corollary 2. *We have* $\mathbf{Q}(q_1, \ldots, q_n) = \mathbf{Q}(\sqrt{q_{1,1}}, \ldots, \sqrt{q_{1,k_1}}, \ldots, \sqrt{q_{n,1}}, \ldots, \sqrt{q_{n,k_n}})$, *and f is non-zero if and only if $f(q_1, \ldots, q_n) \neq 0$.*

Proof. The first equality follows from Corollary 1 by noting the following sandwiched containment

$$\begin{aligned} &\mathbf{Q}(\sqrt{q_{1,1}}, \ldots, \sqrt{q_{1,k_1}}, \ldots, \sqrt{q_{n,1}}, \ldots, \sqrt{q_{n,k_n}}) \\ = \ &\mathbf{Q}(\sum_{i=1}^{n} q_i) \\ \subseteq \ &\mathbf{Q}(q_1, \ldots, q_n) \\ \subseteq \ &\mathbf{Q}(\sqrt{q_{1,1}}, \ldots, \sqrt{q_{1,k_1}}, \ldots, \sqrt{q_{n,1}}, \ldots, \sqrt{q_{n,k_n}}). \end{aligned}$$

Hence for all $j \geq 1$

$$[\mathbf{Q}(q_1, \ldots, q_j) : \mathbf{Q}(q_1, \ldots, q_{j-1})] = 2^{k_j}.$$

The second claim follows from this by a simple induction. □

The algorithm by Chen and Kao [4] uses the special case of Corollary 2 on pair-wise relatively prime integers to arrive at the test for polynomial identities. With Corollary 2, we have generalized their algorithm to be used with any multiplicatively independent rationals or integers. In the next section we estimate the probability that polynomially many random integers are multiplicatively dependent modulo squares. We show that this probability is superpolynomially small. Thus most such sequences of integers are multiplicatively independent. On the other hand, the probability that polynomially many random integers are pair-wise relatively prime is exponentially small.

3 The Probability of Multiplicative Dependence

In this section we consider various probabilities related to multiplicative dependence.

We will first show that the probability that a multiplicative dependence modulo squares among polynomially many random integers is extremely small. For this result we need the function $L(N) := \exp(\sqrt{\log N \log\log N})$. Pomerance [10] showed that if we choose random positive integers $q_1, \ldots, q_n \leq N$, then (from our point of view)

$$\Pr[\deg \mathbf{Q}(\sqrt{q_1}, \ldots, \sqrt{q_n}) < 2^n] \to \begin{cases} 0, & \text{if } n \leq L(N)^{\sqrt{2}-\epsilon}; \\ 1, & \text{if } n \geq L(N)^{\sqrt{2}+\epsilon}. \end{cases}$$

His bound for the probability in the first case is $O(1/\sqrt{\log N})$, which is not sharp enough. We are, however, interested in much smaller n relative to N, and can simplify his argument and still get a useful bound.

Theorem 3. *Fix $d > 0$. Let $q_1, q_2, \ldots, q_n$ be chosen independently from the uniform distribution on $\{1, \ldots, N\}$, with $n \le (\log N)^d$. The probability that there is a nonempty $S \subseteq \{1, \ldots, n\}$ and an integer r such that $\prod_{i \in S} q_i = r^2$ is at most $L(N)^{-1/\sqrt{2}+o(1)}$.*

Proof. We will call a number B-smooth if all its prime factors are less than or equal to B. If a nontrivial product from $q_1, \ldots, q_n$ is a square, then at least one of the following two events must occur. Either for some prime $p > B$, we have $p^2 | q_i$ for some i or $p | q_i, q_j$ for some $i < j$ (call this C), or one of $q_1, \ldots, q_n$ is B-smooth (call this S). We have $\Pr[C] \le \sum_{p>B} \left(n/p^2 + \binom{n}{2}/p^2\right) \le \binom{n+1}{2}/B$. Also, $\Pr[S] \le n\frac{\psi(N,B)}{N}$, where $\psi(N, B)$ counts the B-smooth numbers $\le N$. From [2] we know that when

$$u := \frac{\log N}{\log B} \le \frac{\log N}{\log \log N},$$

we have $\psi(N, B)/N \le e^{-u \log u(1+o(1))}$, uniformly as $u \to \infty$. The choice of B involves a tradeoff that is familiar from the analysis of factorization algorithms: we would like B large so as to make C unlikely, but we would also like it small so as to prevent S. A good choice is

$$B = \exp\left(\sqrt{\frac{\log N \log \log N}{2}}\right), \qquad u = \sqrt{\frac{2 \log N}{\log \log N}},$$

which makes the probabilities roughly equal. Then it is straightforward to prove that $\Pr[C]$ and $\Pr[S] \le L(N)^{-1/\sqrt{2}+o(1)}$, from which the result follows. □

Note that the bound $L(N)^{-1/\sqrt{2}+o(1)}$ is asymptotically much less than inverse exponential polylog in input size $\log N$,

Corollary 3. *For any $c > 0$, under the hypotheses of the last theorem, we have*

$$\Pr[\deg \mathbf{Q}(\sqrt{q_1}, \ldots, \sqrt{q_n}) < 2^n] = o(1/\exp((\log \log N)^c))$$

as $N \to \infty$.

The probability that $q_1, \ldots, q_n$ are pairwise relatively prime is also of interest. It is easy to see that the probability of this is exponentially small, for example by considering adjacent pairs. Below we give a bound with explicit dependence on the parameters.

Theorem 4. *Let $q_1, \ldots, q_n$ be chosen independently from the uniform distribution on $\{1, \ldots, N\}$. The probability that these numbers are pairwise relatively prime is at most $(1 + n)2^{-n} + n/N$.*

Proof. For the q_i to be pairwise relatively prime, it is necessary and sufficient that this condition hold locally for each p. To get our estimate we use only $p = 2$. First assume that N is even. The residues of the q_i mod 2 are a random sequence of bits, of which at most one can be 0. Using the binomial distribution,

the probability of this event is $2^{-n} + n2^{-n} = (1+n)2^{-n}$. If N is odd, either all q_i are $\leq N-1$, for which the previous case applies, or some q_i equals N. The chance of this last event is at most n/N. □

One can get better results by considering more primes.

Theorem 5. *For n fixed and $N \to \infty$, the probability of the last theorem has the limit*

$$P(n) = \prod_p \left\{ \left(1 + \frac{n}{p-1}\right) \left(1 - \frac{1}{p}\right)^n \right\}.$$

Proof. Let E_N be the event that $q_1, \ldots, q_n$ are pairwise relatively prime. Let G_p ("good at p") be the event that no pair has a gcd divisible by p, and B_p ("bad at p") its complement. Then

$$\Pr\left[\bigcap_{p \leq x} G_p\right] - \sum_{p>x} \Pr[B_p] \leq \Pr[E_N] \leq \Pr\left[\bigcap_{p \leq x} G_p\right].$$

We have $\sum_{p>x} \Pr[B_p] \leq \frac{\binom{n}{2}}{x}$, so

$$P_x(n) - \frac{\binom{n}{2}}{x} \leq \liminf_{N\to\infty} \Pr[E_N] \leq \limsup_{N\to\infty} \Pr[E_N] \leq P_x(n),$$

where

$$P_x(n) = \prod_{p \leq x} \left(1 + \frac{n}{p-1}\right) \left(1 - \frac{1}{p}\right)^n.$$

By the binomial theorem, each factor of P_x is $1 + O(p^{-2})$, so $\lim_{x\to\infty} P_x = P(n)$, which implies the theorem. □

4 The Lower Bound

In this section we prove that for any choice of s multiplicatively independent $q_{i,j}$'s of polynomially many bits, where $1 \leq i \leq n$, $1 \leq j \leq k_i = \lceil \log_2(d_i + 1) \rceil$, $s = \sum_{i=1}^n k_i$, and any prespecified choice of signs in the sums $q_i = \sum_{j=1}^{k_i} (-1)^{b_{i,j}} \sqrt{q_{i,j}}$, and any polynomial bit precision L, there are non-zero polynomials $f(x_1, \ldots, x_n)$, where the degree of x_i is at most d_i and the coefficients are integral and all bounded in n, such that while $f(q_1, \ldots, q_n) \neq 0$, for all $\ell \leq L$, $f([q_1]_\ell, \ldots, [q_n]_\ell) = 0$, where $[q_i]_\ell$ denotes the ℓ-bit truncation of q_i. This provides a negative answer to the open question from Chen and Kao [4]. We can even take f multilinear and all coefficients from $\{-1, 0, 1\}$.

In fact, a much stronger result can be shown where L, the bit precision, can be substantially increased from polynomial in n to almost 2^n, yet still some non-zero multilinear polynomial f with coefficients from $\{-1, 0, 1\}$ can vanish on all truncations up to L bits. Such a polynomial can even simultaneously

vanish on all L-bit truncations on a large (almost 2^n) set of arbitrarily chosen initial irrational points. The proof is a direct adaptation of Dirichlet's method showing the existence of simultaneous rational approximations with bounded denominators.

Theorem 6. *Given any n and any degree sequence $d_1, \ldots, d_n$, let $D = \prod_{i=1}^{n}(d_i+1)$. Let C, L, Q be arbitrary positive integers, such that $(2C+1)^D > (2^{L+1}CDQ)^L$. Let $q_{1,1}, \ldots, q_{1,k_1}, \ldots, q_{n,1}, \ldots, q_{n,k_n}$ be any integers multiplicatively independent modulo squares, where $k_i = \lceil \log_2(d_i+1) \rceil$. Let $q_i = \sum_{j=1}^{k_i} a_{i,j} q_{i,j}$, where $a_{i,j}$ are any non-zero integers. Assume $\prod_i |q_i|^{d_i} \leq Q$. Then, there are non-zero polynomials $f(x_1, \ldots, x_n)$ with integral coefficients bounded by C, and degree in x_i bounded by d_i, such that $f([q_1]_\ell, \ldots, [q_n])_\ell) \neq 0$, for all $1 \leq \ell \leq L$, where $[q_i]_\ell$ denotes the ℓ-bit truncation of q_i.*

Corollary 4. *If $2C = 2^c$, $Q = 2^q$, and $2^n \geq L(L+c+q+n)/c$, then there are non-zero multilinear polynomials satisfying the above condition.*

Corollary 5. *If $2^n \geq tL(L+q+n+1)$, then there are non-zero multilinear polynomials with coefficients from $\{-1, 0, 1\}$ that vanish on all L-bit truncations on a set of t arbitrarily chosen initial irrational points $q^{(1)}, q^{(2)}, \ldots, q^{(t)}$, where each $q^{(i)} = (q_1^{(i)}, q_2^{(i)}, \ldots, q_n^{(i)}$ has the form given in Theorem 6.*

Proof. For any $\xi = (\xi_{j_1,\ldots,j_n})$, where $\xi_{j_1,\ldots,j_n}$ is a sequence of integers, with $0 \leq j_i \leq d_i$, $1 \leq i \leq n$ and $|\xi_{j_1,\ldots,j_n}| \leq C$, define a polynomial f_ξ by

$$f_\xi(x_1, \ldots, x_n) = \sum_{0 \leq j_i \leq d_i, 1 \leq i \leq n} \xi_{j_1,\ldots,j_n} x_1^{j_1} \cdots x_n^{j_n}.$$

The evaluations of f at all ℓ-bit truncations of q_i defines a (linear) map from $[-C, C]^D$ to $(-CDQ, CDQ)^L$, by $\xi \mapsto (f_\xi([q_1]_\ell, \ldots, [q_n]_\ell), 1 \leq \ell \leq L)$. Note that there are at most D non-zero terms in f_ξ and each term is strictly less than CQ, being strict since q_i is irrational.

Now divide the range into $2^{L+1}CDQ$ equal parts along every dimension. More precisely, for each dimension $1 \leq i \leq L$, divide the interval range $(-CDQ, CDQ)$ into $2^{L+1}CDQ$ equal length subintervals, i.e., for $1 \leq j_i < 2^{L+1}CDQ$ the j_ith subinterval is $(-CDQ + (j_i - 1)/2^L, -CDQ + j_i/2^L]$ and for the last one $j_i = 2^{L+1}CDQ$, the subinterval is $(CDQ - 1/2^L, CDQ)$. In this way, we obtain $(2^{L+1}CDQ)^L$ cells each congruent to $(0, \frac{1}{2^L}]^L$ (except on each dimension the last one has open boundaries on both sides).

Since $(2C+1)^D > (2^{L+1}CDQ)^L$, *some* two images fall in the same cell, i.e., there are two distinct tuples $\xi \neq \xi'$ such that

$$|f_\xi([q_1]_\ell, \ldots, [q_n]_\ell) - f_{\xi'}([q_1]_\ell, \ldots, [q_n]_\ell)| < \frac{1}{2^L},$$

for all $1 \leq \ell \leq L$.

However, each quantity is a rational number with at most L bits after the decimal point. In other words, both $2^L f_\xi([q_1]_\ell, \ldots, [q_n]_\ell)$ and $2^L f_{\xi'}([q_1]_\ell, \ldots, [q_n]_\ell)$ are integers. It follows that $f_\xi([q_1]_\ell, \ldots, [q_n]_\ell) - f_{\xi'}([q_1]_\ell, \ldots, [q_n]_\ell) = 0$, for all $1 \leq \ell \leq L$. Now take $z_{j_1,\ldots,j_n} = \xi_{j_1,\ldots,j_n} - \xi'_{j_1,\ldots,j_n}$, then the non-zero polynomial f_z has $f_z([q_1]_\ell, \ldots, [q_n]_\ell) = 0$, for all $1 \leq \ell \leq L$. □

Acknowledgement

We thank Zhi-Zhong Chen for describing to me his result with Ming-Yang Kao during his visit to the U.S. and for interesting discussions on the problem. We also thank Osamu Watanabe for helpful discussions of the problem.

References

1. E. Bach, J. Driscoll, and J. Shallit. Factor Refinement, Journal of Algorithms 15 (1993) 199-222.
2. E. R. Canfield, P. Erdős, and C. Pomerance. On a problem of Oppenheim concerning "factorisatio numerorum." J. Number Theory **17**, 1983, 1-28.
3. S. Chari, P. Rohatgi and A. Srinivasan. Randomness-optimal unique element isolation with application to perfect matching and related problems. *SIAM J. on Computing*, 24 (1995), 1036–1050.
4. Z. Chen and M. Kao. Reducing Randomness via Irrational Numbers In *Proceedings of the 29th ACM STOC*, pages 200–209. 1997.
5. Z. Chen. Personal communications.
6. S. Lang. *Algebra*. Addison-Wesley, 1971.
7. L. Lovász. On determinants, matchings and random algorithms. In L. Budach (ed), *Fundamentals of Computing Theory*. Akademia-Verlag, Berlin, 1979.
8. G. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13: 300–317, 1976.
9. N. Nisan. Extracting Randomness: How and Why–A survey. In the Proc. of *Computational Complexity*, pp 44-58. 1996.
10. C. Pomerance. Multiplicative independence for random integers. In *Analytic Number Theory*, Volume 2: Proceedings of a Conference in Honor of Heini Halberstam, ed. B. C. Berndt et al., Birkhäuser, Boston, 1996, pp. 703-711.
11. M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12 : 128–138, 1980.
12. J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. Journal of ACM, 27 (1980), 701-717.
13. R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6 (1): 84–85, March 1977. See also *SIAM Journal on Computing*, 7, 1 Feb 1978, 118.
14. R. Zippel. Probabilistic algorithms for sparse polynomials. In the Proc. of *EUROSAM 79*, Lecture Notes in Computer Science, vol. 72, Springer-Verlag, 1979, 216–226.
15. D. Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, Vol. 16, pages 367–391, 1996.

A Randomized Algorithm for Gossiping in Radio Networks

Marek Chrobak[1], Leszek Gąsieniec[2], and Wojciech Rytter[3]

[1] Department of Computer Science, University of California, Riverside, CA 92521. marek@cs.ucr.edu.
[2] Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK. leszek@csc.liv.ac.uk.
[3] Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02–097, Warszawa, Poland, and Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK. rytter@csc.liv.ac.uk.

Abstract. We present an $O(n \log^4 n)$-time randomized algorithm for gossiping in radio networks with unknown topology. This is the first algorithm for gossiping in this model whose running time is only a poly-logarithmic factor away from the optimum. The fastest previously known (deterministic) algorithm for this problem works in time $O(n^{3/2} \log^2 n)$.

1 Introduction

The two classical problems of disseminating information in computer networks are *broadcasting* and *gossiping*. In broadcasting, we want to distribute a given message from a distinguished source node to all other nodes in the network. In gossiping, each node v in the network initially contains a message m_v, and we wish to distribute each message m_v to all nodes in the network. In both problems, we would like to minimize the time needed to complete the task.

In radio networks, a message transmitted by a processor is sent to all processors within its range. The range relation is represented by a graph of nodes with directed edges between them. All processors work synchronously, and if a processor u transmits a message m at time step t, the message reaches each neighbor v of u at the same time step. Node v will successfully receive m only if u is the only processor, among those whose range contains v, that transmits at time t. Since the communication links are uni-directional, there is no feedback mechanism in the network (see, for example, [20]), and thus, in general, a node does not know for certain whether its transmissions were successful. Further, we assume that collisions cannot be resolved nor detected, that is, if messages from two or more processors reach v at time t, v does not get any message and it does not know that the collision occurred. This is motivated by situations when message collisions are difficult to distinguish from background noise, in which protocols that do not depend on the accuracy of the collision detection mechanism (see [13,12]) will be more reliable.

We focus on gossiping algorithms that do not use any information about network topology. Such topology-independent algorithms are useful in networks

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 483–492, 2001.
© Springer-Verlag Berlin Heidelberg 2001

with mobile users or unstable topologies, since then one does not need to change or reconfigure the protocol after topology changes. As long as the network is strongly connected and no changes occur during the actual execution of the algorithm, the task of gossiping will complete successfully. The strong connectivity assumption is necessary for gossiping to be meaningful.

Past work. Most of the previous work on radio networks focussed on broadcasting. If the topology of the network is known to all processors, Gaber and Mansour [14] showed that broadcasting can be achieved in time $O(D + \log^5 n)$, where D is the network diameter. Diks *et al* [10] gave efficient broadcasting algorithms for special types of known networks. It is also known that computing an optimal broadcast schedule for a given network is NP-hard, even for points in the plane, where the graph is induced by node ranges, see [8,26].

For networks with unknown topology, Bar-Yehuda *et al* [3] gave a randomized algorithm that achieves broadcast in expected time $O(D \log n + \log^2 n)$. This is very close to the lower bound of $\Omega(D \log(n/D))$, by Kushilevitz and Mansour [18], and it matches this lower bound for a wide range of depth values, for example when $D = \Theta(n^{1-\epsilon})$, for any $\epsilon > 0$. Further, if D is a constant, it also matches the lower bound of $\Omega(\log^2 n)$ for constant diameter networks, obtained by Alon *et al* [1].

In the deterministic case, Bar-Yehuda *et al* [3] gave an $\Omega(n)$ lower bound for constant diameter networks. For general networks, the best currently known lower bound of $\Omega(n \log n)$ was obtained by Bruschi and del Pinto [5], and, independently by Chlebus *et al* [6]. In [6], the authors also present a broadcast algorithm with time complexity $O(n^{11/6})$ – the first sub-quadratic upper bound. This upper bound was later improved to $O(n^{5/3} \log^3 n)$ by De Marco and Pelc [11]. Chlebus *et al* [7] developed several broadcasting algorithms, including one with time complexity $O(n^{3/2})$. In an independent work, using a probabilistic construction, Peleg [23] gave an $O(n^{3/2}\sqrt{\log n})$ upper bound. Recently, Chrobak, Gąsieniec and Rytter [9] presented a deterministic algorithm for broadcasting with time complexity $O(n \log^2 n)$, thus nearly matching the lower bound of $\Omega(n \log n)$ from [5,6]. As in [23], this broadcasting algorithm was constructed using a probabilistic argument.

The problem of gossiping has been intensely studied in various network models (see, for example, [15]). For radio networks, Ravishankar and Singh [24,25] studied gossiping algorithms for some restricted topologies, including paths and rings, under probabilistic assumptions on the spatial distribution of nodes. In our previous work, [9], we developed a deterministic algorithm for gossiping with time complexity $O(n^{3/2} \log^2 n)$, which, to our knowledge, is the only sub-quadratic algorithm for gossiping in radio networks with unknown topology.

Our results. In this paper we give a randomized $O(n \log^4 n)$-time algorithm for gossiping in radio networks with unknown topology. Our basic algorithm is Monte Carlo and it has the following performance characteristics: for any $0 < \epsilon < 1$, in time $O(n \log^3 n \log(n/\epsilon))$ it achieves broadcast with probability at least $1 - \epsilon$. This easily yields a Las Vegas algorithm with expected running time $O(n \log^4 n)$.

2 Preliminaries

Radio networks. A *radio network* (see [3,7]) is defined as an n-node directed graph whose nodes are assigned unique identifiers from the set $\{1, 2, \ldots, n\}$. Throughout the paper, for gossiping to be meaningful, we assume that the network is strongly connected. If there is an edge from u to v, then we say that v is an *out-neighbor* of u and u is an *in-neighbor* of v.

Initially, each node v contains a message m_v and has no other information. The time is divided into discrete time steps. All nodes start simultaneously, have access to a common clock, and work synchronously. (As noted by Peleg [23], the assumption about a common clock is not necessary.) At any time step, a node can be in one of two states: the *receiving state* or the *transmitting state*. A gossiping algorithm is a protocol that for each identifier *id* and for each time step t, given all past messages received by *id*, specifies the state of *id* at time t. If *id* transmits at time t, the protocol specifies the message. A message m transmitted at time t from a node u is sent instantly to all its out-neighbors. However, an out-neighbor v of u receives m at time step t only if v is in the receiving state and if no collision occurred, that is, if the other in-neighbors of v do not transmit at time t at all. Further, collisions cannot be distinguished from background noise. If v does not receive any message at time t, it knows that either none of its in-neighbors transmitted at time t, or that at least two did, but it does not know which of these two events occurred.

The running time of a gossiping algorithm is the smallest t such that for any strongly connected network topology, and for any assignment of identifiers to the nodes, each node receives all messages m_v no later than at step t.

Simplifying assumptions. For clarity of presentation, we will present our algorithms as if the nodes knew n, the size of the network. This assumption can be eliminated by a standard doubling technique (see [6,7]) that works as follows: We organize the computation into phases, and we modify a given algorithm so that in phase i only nodes with labels at most 2^i participate in the algorithm. This does not change the asymptotic running time.

Further, we will also assume throughout the paper that n is a power of 2. For other n, the processors can execute the algorithm for the nearest power of 2 larger than n, without changing the asymptotic running time.

Notation. By V we denote the set of nodes, and individual nodes are denoted by letters $u, v, \ldots$. Messages are denoted by letter m, possibly with indices. The message originating from a node v is denoted by m_v. The whole set of initial messages is $M = \{m_v : v \in V\}$. During the computation, each node v will store a set of messages M_v that have been received by v so far. Initially, $M_v = \{m_v\}$. Without loss of generality, whenever a node is in the transmit mode, we can assume that it transmits the whole M_v. This is achieved by a procedure denoted *transmit*(M_v). Procedure *receive*() puts v in the receive mode and it returns the received message, or *null* if no message has been received.

3 Limited Broadcast

One component of our algorithm is a procedure for *limited broadcasting*. Given an integer k and a node v, the goal of limited broadcasting is to send the message M_v to at least k nodes in the network. We refer to v as the *source node* or the node that *initiates* the broadcast, and to M_v as the *source message*.

In [9], the broadcasting algorithm is defined by a sequence $\bar{S} = S_0 S_1 \ldots$ of subsets of $\{1, 2, \ldots, n\}$. At time t, any node v that has already received the source message checks whether $v \in S_t$. If so, v transmits the message, otherwise v is quiet. We modify the algorithm from [9], so that it performs the broadcasting procedure for only $O(k \log^2 n)$ steps. Below, we appropriately refine the correctness proof, since the proof from [9] is not sufficient for our purpose.

The pseudo-code for the algorithm executed by each node is given below. Each node v has a boolean flag $active_v$ that indicates whether v is *active*, that is, whether v has received a source message. Each iteration of the for-loop lasts one time step. The value of constant γ will be determined later.

Procedure LtdBroadcast$_v(k)$.

for $\tau = 0, 1, \ldots, \gamma k \log^2 n - 1$ **do**
  if $active_v$ and $v \in S_\tau$ **then** *transmit*(M_v)
  else
    $m \leftarrow$ *receive*();
    if $m \neq$ *null* **then**
      $M_v \leftarrow M_v \cup \{m\}$;
      $active_v \leftarrow$ *true*

Lemma 1. *Assume that initially exactly one node u is active and that all nodes v begin executing* LtdBroadcast$_v(k)$ *simultaneously. Then, for some constant γ independent of k, after the computation is complete at least k nodes will receive the message from u.*

Proof. Without loss of generality, assume $n \geq 2$. In [9], it was shown that there is a constant α such that for each $j = 0, \ldots, \log n$ there is a family $\bar{S}_j = (S_{j,0}, S_{j,1}, \ldots, S_{j,a_j - 1})$ of $a_j = \alpha 2^j \log n$ sets with the following property:

(∗) For any two disjoint sets X, Y with $2^{j-1} \leq |X| \leq 2^j$ and $|Y| \leq 2^j$ there exists a set $S_{j,i}$ in $\bar{S}_j$ such that $|X \cap S_{j,i}| = 1$ and $Y \cap S_{j,i} = \emptyset$.

Let $\gamma = 12\alpha + 2$. The sequence $\bar{S}$ consists of *stages*, with each stage, except possibly the last one, having $\log n + 1$ steps. Note that the number of stages is $\lceil \gamma k \log^2 n / (\log n + 1) \rceil \geq 6\alpha k \log n + 1$. The transmission set at the jth step of stage s, that is S_τ for $\tau = s(\log n + 1) + j$, is $S_{j, s \bmod a_j}$.

Among the active nodes we distinguish two types of nodes: *frontier nodes*, which still have inactive out-neighbors, and *inner nodes*, which don't. If there is a time step $t < \gamma k \log^2 n$ when there are k or more frontier nodes, then the lemma holds trivially. So from now on we assume that at each step the number of frontier nodes is less than k.

We define a sequence of stages $s_0 = 0, s_1, \ldots, s_{l+1} = 6\alpha k \log n$, where $1 \leq s_{c+1} - s_c \leq 2\alpha k \log n$ for all c. Denote by i_c and f_c the number of inner and frontier nodes when stage s_c is about to start. We will choose $s_1, \ldots, s_l$ so that the following invariant holds for each $c \leq l$:

$$2i_c + f_c \geq \frac{s_c}{2\alpha \log n}. \tag{1}$$

Given (1), we can prove the lemma as follows. The number of nodes that have received the message when stage s_{l+1} ends is at least

$$\begin{aligned} i_l + f_l &\geq \tfrac{1}{2}(2i_l + f_l) \\ &\geq \frac{s_l}{4\alpha \log n} \\ &\geq \frac{s_{l+1} - 2\alpha k \log n}{4\alpha \log n} \\ &= k. \end{aligned}$$

So it is sufficient to construct $s_1, \ldots, s_l$ that satisfy (1). We define these stages inductively. For $c = 0$ we have $s_0 = 0$, $i_0 = 0$ and $f_0 = 1$, so (1) holds. Suppose we have determined some s_c. If $s_c > 4\alpha k \log n$, set $l = c$ and we are done. Otherwise, we proceed as follows.

Let F be the set of frontier nodes at the beginning of stage s_c, and let g be such that $2^{g-1} \leq |F| < 2^g$. For each $j = 1, \ldots, g$, let Y_j be the set of nodes that received the message in stages $s_c, s_c + 1, \ldots, s_c + a_j - 1$ (but were inactive when stage s_c started).

We have two sub-cases:

Case 1: There is j for which $|Y_j| \geq 2^j$. In this case, take $s_{c+1} = s_c + a_j$. At least $|Y_j|$ new nodes received the message, so

$$\begin{aligned} 2i_{c+1} + f_{c+1} &\geq 2i_c + f_c + |Y_j| \\ &\geq \frac{s_c}{2\alpha \log n} + 2^j \\ &\geq \frac{s_{c+1}}{2\alpha \log n}. \end{aligned}$$

Case 2: For each j we have $|Y_j| \leq 2^j$. We show that in this case all nodes in F will become inner after a_g stages.

Fix any node v that is inactive when stage s_c starts, and whose set X of in-neighbors in F is not empty. Pick j such that $2^{j-1} \leq |X| < 2^j$. Since $|Y_j| \leq 2^j$, by property $(*)$, family $\tilde{S}_j$ contains a set $S_{j,i}$ that hits X and avoids Y_j. This $S_{j,i}$ will occur in one of the stages $s_c, s_c + 1, \ldots, s_c + a_j - 1$.

All in-neighbors of v are either in X or are inactive when stage s_c starts. When we use $S_{j,i}$ for transmission then:

(i) exactly one in-neighbor of v in X will transmit because $|S_{j,i} \cap X| = 1$,
(ii) the nodes from Y_j will not interfere because $S_{j,i} \cap Y_j = \emptyset$,

(iii) the nodes that were inactive at the beginning of stage s_c and are not in Y_j remain inactive when $S_{j,i}$ is issued, so they will not transmit.

Therefore v will receive the message when $S_{j,i}$ is issued (unless it has already received it earlier). Since v was an arbitrary inactive out-neighbor of F, we conclude that all nodes in F will become inner after a_g stages.

Take $s_{c+1} = s_c + a_g$. In this case $i_{c+1} \geq i_c + f_c$ and $f_c \geq 2^{g-1}$, so

$$\begin{aligned} 2i_{c+1} + f_{c+1} &\geq 2i_c + f_c + f_c \\ &\geq \frac{s_c}{2\alpha \log n} + 2^{g-1} \\ &\geq \frac{s_{c+1}}{2\alpha \log n}. \end{aligned}$$

We thus proved that (1) holds for s_{c+1}. Further, since $|F| \leq k$, we have $a_g = \alpha 2^g \log n \leq 2\alpha k \log n$, and thus in both sub-cases we have $s_{c+1} - s_c \leq 2\alpha k \log n$. The proof of the lemma is now complete.

4 Distributed Coupon Collection

In each phase of our algorithm we will attempt to distribute each message m_v to some number of nodes, by performing a sequence of limited broadcasts. We need to achieve two goals. To obtain a good running time, the number of limited broadcasts must be small. Further, each message m_v should participate in at least one limited broadcast, that is, m_v must be in at least one M_u, for some u that initiates a limited broadcast.

To choose the nodes v for which we initiate a limited broadcast, we use randomization. The principle behind the random process we use is similar to that in the coupon collector's problem. There are two differences though. First, each coupon may have several copies. Second, since we do not have enough time to coordinate the choices, we cannot guarantee that exactly one node will initiate broadcasting.

We think of V as a set of n bins and M as a set of n coupons. Each coupon has at least k copies, each copy belonging to a different bin. M_v is the set of coupons in bin v. Consider the following process: At each step, we open bins at random, by choosing each bin, independently, with probability $1/n$. If exactly one bin, say v, is opened, all coupons from M_v are *collected*. If no bin is opened, or if two or more bins are opened, a failure occurs and no coupons are collected. How many steps do we need so that with high probability (a copy of) each coupon is collected?

The distributed coupon collection can be written in pseudocode as follows.

Procedure *DistCouponColl*(s)**.**

repeat s **times**
  for each bin v **do**
    with probability $1/n$ **do** open v
      else close v
  if exactly one bin v is opened **then**
    collect all coupons from M_v

Lemma 2. *Assume we have n bins and n coupons, and that each coupon has at least k copies, each copy belonging to a different bin. Let δ be a given constant, $0 < \delta < 1$, and $s = (4n/k)\ln(n/\delta)$. Then, after performing DistCouponColl(s), with probability at least $1 - \delta$, all coupons will be collected.*

Proof. The lemma is trivially true for $n = 1$, so we can assume that $n \geq 2$. Let $\mathcal{X}_{m,j}$ be the event that coupon m is collected at a given step j. Then $\Pr[\,\mathcal{X}_{m,j}\,]$ is the probability that one bin containing m is opened and all other bins are closed.
For all m and j we have

$$\begin{aligned}\Pr[\,\mathcal{X}_{m,j}\,] &\geq \frac{k}{n}\left(1-\frac{1}{n}\right)^{n-1} \\ &\geq \frac{k}{n}\left(1-\frac{1}{n}\right)^{n} \\ &\geq \frac{k}{4n},\end{aligned}$$

where the last inequality follows from the fact that the sequence $(1 - 1/n)^n$ is monotonely increasing. The probability that some m is not collected in s steps is

$$\begin{aligned}\Pr\Big[\,\bigvee_{m\in M}\bigwedge_{j\leq s}\neg\mathcal{X}_{m,j}\,\Big] &\leq n\left(1-\frac{k}{4n}\right)^{s} \\ &\leq ne^{-sk/4n} \\ &\leq \delta,\end{aligned}$$

by the definition of s.

5 The Gossiping Algorithm

We now present our Monte Carlo algorithm for gossiping. Each node v performs its version of the algorithm.

Algorithm RANDGOSSIP$_v(\epsilon)$**.**

$\delta \leftarrow \epsilon / \log n$
for $i = 0, 1, \ldots, \log n - 1$ **do**
Phase$i$:
$s_i \leftarrow (4n/2^i) \ln(n/\delta)$
repeat s_i **times**
for each node v **do** (in parallel)
with probability $1/n$ **do** $active_v \leftarrow true$
else $active_v \leftarrow false$
LTDBROADCAST$_v(2^{i+1})$

Theorem 1. *Let ϵ, $0 < \epsilon < 1$, be a given constant. With probability at least $1-\epsilon$, Algorithm* RANDGOSSIP(ϵ) *completes gossiping in time $O(n \log^3 n \log(n/\epsilon))$.*

Proof. In phase i, the call to LTDBROADCAST$_v(2^{i+1})$ costs $O(2^i \log^2 n)$, so phase i costs $O(s_i 2^i \log^2 n) = O(n \log^2 n \log(n/\delta))$. Since we have $\log n$ phases and $\delta = \epsilon / \log n$, this implies the bound on the running time of the algorithm.

Initially, when phase 0 starts, each m_u is in one set M_v, namely in M_u. The algorithm attempts to maintain the invariant that after phase i each m_u is in at least 2^{i+1} sets M_v. If this invariant is preserved at each phase, the gossiping will complete successfully, since after phase $\log n - 1$ each m_u will be in n sets M_v. Thus it is sufficient to prove that the probability that the invariant fails in some phase is at most ϵ.

The process of distributing messages in a given phase i is equivalent to the distributed coupon collection problem described in the previous section, where we view each node as a bin, and active nodes correspond to open bins. Thus, by Lemma 2, the probability that the invariant fails in this phase, assuming that it has not failed in any previous phase, is at most δ. So, overall, the probability of failure in some phase is at most $\log n \cdot \delta = \epsilon$.

To obtain a Las Vegas algorithm, run RANDGOSSIP(ϵ) with $\epsilon = 1/n$. After the algorithm halts, each node that has not received n different initial messages, announces that a failure occurred. Since all these nodes send out the same message, this can be achieved with just one broadcast. If B is the running time of the broadcasting algorithm, after B steps each node knows whether the gossiping was successful or not. If the gossiping failed, we run a naive, deterministic ROUNDROBIN algorithm that consists of n rounds, with each node transmitting once in each round. This will achieve gossiping in time $O(n^2)$. Overall, the expected running time will be $O((1 - 1/n) n \log^4 n + (1/n) n^2) = O(n \log^4 n)$. Concluding, we get the following theorem.

Theorem 2. *There exists a randomized Las Vegas algorithm for gossiping with running time $O(n \log^4 n)$.*

6 Final Comments

Several open problems remain. The only known lower bounds for gossiping are those for broadcasting [1,5,7], a seemingly easier problem. The gap between lower and upper bounds is particularly wide in the deterministic case: between $\Omega(n \log n)$ and $O(n^{3/2} \log^2 n)$. Closing or at least reducing this gap is an interesting open problem.

Our algorithm is probably not optimal. One possible research direction is to investigate whether one can improve the running time of our algorithm by using the randomized broadcasting algorithm from [3] to perform limited broadcast. It is not quite clear whether the algorithm from [3] can be modified to satisfy Lemma 1. Further, the analysis of this modified algorithm will probably be much more complicated.

Improving the bounds on gossiping or broadcasting may resolve the question whether gossiping is harder than broadcasting in the radio network model (both in the randomized and the deterministic case). Our result implies that, at least in the randomized case, the difference is at most a poly-logarithmic factor.

Acknowledgements

This work was supported by grant EPSRC GR/N09077. M.Chrobak's research was also partially supported by NSF grant CCR-9988360.

References

1. N. Alon, A. Bar-Noy, N. Linial and D. Peleg, A lower bound for radio broadcast, *Journal of Computer and System Sciences* 43 (1991) 290–298.
2. N. Alon, A. Bar-Noy, N. Linial and D. Peleg, Single round simulation of radio networks, *Journal of Algorithms* 13 (1992) 188–210.
3. R. Bar-Yehuda, O. Goldreich, and A. Itai, On the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization, *Journal of Computer and System Sciences* 45 (1992) 104–126.
4. R. Bar-Yehuda, A. Israeli, and A. Itai, Multiple communication in multi-hop radio networks, *SIAM Journal on Computing* 22 (1993) 875–887.
5. D. Bruschi and M. del Pinto, Lower bounds for the broadcast problem in mobile radio networks, *Distributed Computing* 10 (1997) pp. 129-135
6. B.S. Chlebus, L. Gąsieniec, A.M. Gibbons, A. Pelc, and W. Rytter, Deterministic broadcasting in unknown radio networks, in *Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, California, 2000, pp. 861–870.
7. M. Chlebus, L. Gąsieniec, A. Östlin and J.M. Robson, Deterministic broadcasting in radio networks, *Proc. 27th International Colloquium on Automata, Languages and Programming,* ICALP'2000.
8. I. Chlamtac and S. Kutten, On broadcasting in radio networks - problem analysis and protocol design, *IEEE Transactions on Communications* 33 (1985) 1240–1246.
9. M. Chrobak, L. Gąsieniec and W. Rytter, Fast broadcasting and gossiping in radio networks, 41*st Annual IEEE Conference on Foundations of Computer Science, FOCS'00*, 2000, pp. 575–581.

10. K.Diks, E.Kranakis, D.Krizanc, A.Pelc, The impact of knowledge on broadcasting time in radio networks, In *Proc. 7th European Symp. on Algorithms* (1999) 41-52
11. G. De Marco and A. Pelc, Faster broadcasting in unknown radio networks, unpublished manuscript, 1999.
12. A. Ephremides and B. Hajek, Information theory and communication networks: an unconsummated union, *IEEE Trans. on Inf. Theory*, IT-44 (1998) 2416-2434.
13. R. Gallager, A perspective on multiaccess communications, *IEEE Trans. on Inf. Theory*, IT-31 (1985) 124-142.
14. I. Gaber and Y. Mansour, Broadcast in radio networks, in *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1995, pp. 577–585.
15. S.T. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman, A survey of gossiping and broadcasting in communicatio nnetworks, *Networks* 18 (1988) 319–349.
16. J. Komlos and A.G. Greenberg, An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels, *IEEE Trans. on Inf. Theory*, IT-31 (1985) 302-306.
17. E. Kranakis, D. Krizanc and A. Pelc, Fault-tolerant broadcasting in radio networks, in *Proc. 6th European Symposium on Algorithms*, Venice, Italy, 1998, Springer LNCS 1461, pp. 283–294.
18. E. Kushilevitz and Y. Mansour, An $\Omega(D \lg(N/D))$ lower bound for broadcast in radio networks, *SIAM Journal on Computing* 27 (1998) 702–712.
19. E. Kushilevitz and Y. Mansour, Computation in noisy radio networks, in *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1998, pp. 236–243.
20. J.L. Massey and P. Mathys, The collision channel without feedback, *IEEE Trans. on Inf. Theory*, IT-31 (1985) 192-204.
21. R. Motwani and P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
22. K. Pahlavan and A. Levesque, *Wireless Information Networks*, Wiley-Interscience, New York, 1995.
23. D. Peleg, Deterministic radio broadcast with no topological knowledge, manuscript, 2000.
24. K. Ravishankar and S. Singh, Asymptotically optimal gossiping in radio networks, *Discr. Appl. Math.* 61 (1995) 61–82.
25. K. Ravishankar and S. Singh, Gossiping on a ring with radios, *Parallel Processing Letters* 6 (1996), p. 115-126.
26. A. Sen and M. L. Huson, A new model for scheduling packet radio networks, in *Proc. 15th Ann. Joint Conference of the IEEE Computer and Communication Societies*, 1996, pp. 1116–1124.

Deterministic Application of Grover's Quantum Search Algorithm

Kyoichi Okamoto and Osamu Watanabe

Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo 152-8552, Japan
watanabe@is.titech.ac.jp

Abstract. Grover's quantum search algorithm finds one of t solutions in N candidates by using $(\pi/4)\sqrt{N/t}$ basic steps. It is, however, necessary to know the number t of solutions in advance for using the Grover's algorithm directly. On the other hand, Boyer etal proposed a randomized application of Grover's algorithm, which runs, on average, in $O(\sqrt{N/t})$ basic steps (more precisely, $(9/4)\sqrt{N/t}$ steps) without knowing t in advance. Here we show a simple (almost trivial) deterministic application of Grover's algorithm also works and finds a solution in $O(\sqrt{N/t})$ basic steps (more precisely, $(8\pi/3)\sqrt{N/t}$ steps) on average.

1 Introduction

Grover [Gro96, Gro97] proposed a quantum algorithm — Grover's search algorithm — that solves the following general search problem much faster than any randomized/deterministic algorithm designed on classical computers.

Search Problem
Given: For any $n > 0$, an oracle Boolean function f on the set $\{0,1\}^n$ of binary sequences of length n.
Question: Find some sequence $x \in \{0,1\}^n$ such that $f(x) = 1$.

Remark. In general, a solution, i.e., a sequence $x \in \{0,1\}^n$ satisfying $f(x) = 1$, is not unique, in which case it is sufficient to output *any one* of such solutions.

For any given n, since there are $N = 2^n$ binary strings in $\{0,1\}^n$, it is (almost obvious) that any algorithm solving the above problem needs N steps to find the desired sequence. Surprisingly, though, Grover's search algorithm finds the desired sequence in $O(\sqrt{N})$ quantum computation steps, where each quantum step (which we refer *G-steps*) can be implemented by some poly(n) number of basic quantum gates. More precisely, when there are t solutions, i.e., t binary sequences satisfying f, Grover's algorithm finds some of them in $O(\sqrt{N/t})$ G-steps. Note, however, that one needs to know the number t in advance in order to achieve this better bound; but t, the number of solutions, is usually unknown in advance. For solving this problem, Boyer etal [BBHT96] proposed an algorithm that applies Grover's algorithm with randomly chosen parameters, that

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 493–501, 2001.
© Springer-Verlag Berlin Heidelberg 2001

is, a randomized application of Grover's algorithm, which runs, on average, in $O(\sqrt{N/t})$ G-steps without knowing t in advance. Here we show that a simple (in fact, almost trivial) deterministic application of Grover's algorithm also works and finds a solution in $O(\sqrt{N/t})$ G-steps on average without knowing t in advance. (Notice also here that any classical computation including randominzed one can be simulated by some quantum computation. In fact, for any classical algorithm using any quantum algorithm(s), one can design a single quantum Turing machine that executes the whole algorithm. In this sense, there is no merit of removing randomness from the application algorithm. Yet, the authors think that the simpler algorithm is the better.)

Grover's Algorithm and Its Randomized Application

Let us discuss our problem specifically. We start with recalling Grover's algorithm and some basic facts about the algorithm. Notions and notations for quantum computation we use here are standard; see, e.g., [BV97, Gru99, Hos99, For00].

Consider any $n > 0$ and any oracle function f on $\{0,1\}^n$ for specifying the above general search problem, and let us fix them in the following discussion. That is, our task is to find a sequence $x \in \{0,1\}^n$ that satisfies $f(x) = 1$. A sequence $x \in \{0,1\}^n$ satisfying $f(x) = 1$ is simply called a *solution*. Let $N = 2^n$, i.e., the total number of sequences in $\{0,1\}^n$, and let t denote the number of all solutions among N candidates.

In Grover's algorithm, each sequence $x \in \{0,1\}^n$ corresponds to a quantum base state $|x\rangle$ consisting of n qubits. (In the following, we identify an n qubit base state with the corresponding n bit binary sequence.) The main ingredients of Grover's algorithm are the following three unitary transformations on n qubit states.

Walsh-Hadmard transformation:

$$W : |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$$

(Here $x \cdot y$ denotes the bit-wise inner product.)

Sign flipping on f:

$$S_f : |x\rangle \mapsto (-1)^{f(x)} |x\rangle$$

Sign flipping on 0:

$$S_0 : |0\rangle \mapsto -|0\rangle \text{ and } |x\rangle \mapsto |x\rangle \text{ (if } x \neq 0^n)$$

By using these transformations, one G-step is defined as the following unitary transformation U.

$$U = -WS_0WS_f$$

That is, *one G-step* is to apply this U to a current state. Grover's algorithm is to apply U for some appropriate number of times to the following initial state ϕ_0.

$$|\psi_0\rangle \;=\; \sum_{a\in\{0,1\}^n} \frac{1}{\sqrt{N}}|a\rangle.$$

Formally, by *the j G-step execution of Grover's algorithm* (or more simply $G(j)$) we mean to apply $\underbrace{UUU\cdots U}_{j}$ to ϕ_0 and then observe the obtained state. (We assume that the observation is made so that some n qubit base state (i.e., n bit sequence) is observed with the probability that is the square of its amplitude.)

For justifying this procedure, the following property of U plays a key role [Gro96, Gro97].

Lemma 1. *Consider the quantum state obtained by applying U to ϕ_0 for j times. Then each solution state, i.e., a base state corresponding to a solution, has the same amplitude (which we denote a_j) while the other base state also has the same amplitude (which we denote b_j). Furthermore, by using θ that satisfies $\sin\theta = \sqrt{t/N}$, these amplitudes are stated as follows.*

$$a_j = \frac{1}{\sqrt{t}}\sin((2j+1)\theta),$$
$$b_j = \frac{1}{\sqrt{N-t}}\cos((2j+1)\theta).$$

The angle $(2j+1)\theta$ determines the amplitude a_j. In the following we call this angle *the angle* (*of a solution*) *after applying U for j times* (*to the initial state*) (or more simply, *the angle after executing $G(j)$*). Note that after applying U for $\lfloor(\pi/4)\theta\rfloor$ times, the angle gets close to $\pi/2$; hence, the amplitude of each solution is close to $1/\sqrt{t}$, which means that the total probability that solution states are observed is close to 1. Note also that $(\pi/4)\theta$ is approximately $(\pi/4)\sqrt{N/t}$ by using the approximation $\theta \approx \sin\theta$ $(= \sqrt{t/N})$. This argument leads us to the following theorem of Grover [Gro96, Gro97].

Theorem 1. *Define m_0 by*

$$m_0 \;=\; \left\lfloor \frac{\pi}{4\theta} \right\rfloor \quad \left(\approx\; \frac{\pi}{4}\sqrt{\frac{N}{t}}\right).$$

Then a state observed by the execution of $G(m_0)$ is one of the solutions with probability approximately $1 - 1/N$ (≈ 1).

Therefore, if we can compute m_0, we would simply execute $G(m_0)$ to get some solution. It is, however, not so easy to compute it because t is usually unknown in advance. Note that we cannot simply execute $G(m)$ with some $m > m_0$; in this case, the probability that some solution is observed could become much smaller. In order to solve this problem, Boyer etal [BBHT96] proposed the following randomized algorithm.

It is clear that this algorithm finds a solution. On the other hand, we can show the following time bound.

Algorithm Randomized-Grover
$\lambda \leftarrow 6/5$; $i \leftarrow 0$;
for $i \leftarrow 1$ **to** ∞ **do**
$m \leftarrow \lambda^i$;
select j, $0 \leq j \leq m-1$, uniformly at random;
execute $G(j)$ and let x be the observed state;
if $f(x) = 1$ **then** output x and halt;
end-for;

Fig. 1. A randomized application of Grover's algorithm

Theorem 2. *The average number of G-steps executed in the above algorithm is at most* $(9/4)\sqrt{N/t}$.

Remark. *In the proof of this theorem and in the following discussion in general, we often assume that* t/N *is small enough, say,* $t/N < 1/100$ *and use some approximations like* $\sqrt{t/N} \approx \sin\sqrt{t/N}$ *under this assumption. Thus, our bounds may not be precise; but even so, it is easy to see that the bounds shown here certainly hold by changing constant factors. We leave the tedious precise calculation to the interest reader. Also though our arguments need an assumption such as* $t < N/100$*, this assumption is not essential for designing a fast search algorithm. In fact, if* $t \geq N/100$*, then we can search for a solution by simply picking any n bit sequence randomly, say, 1,000 times.*

2 Deterministic Application of Grover's Algorithm

Here we show that a simple and *deterministic* execution of Grover's algorithm still yields a similar result. More specifically, we consider the algorithm given in Figure 2.

Algorithm Deterministic-Grover
% The parameter k is fixed to 2.
for $i \leftarrow 1$ **to** ∞ **do**
$m \leftarrow 2^i$;
execute $G(m)$ for k times and let x_1, ..., x_k be the observed states;
if $f(x_u) = 1$ for some u, $1 \leq u \leq k$ **then** output x_u and halt;
end-for;

Fig. 2. A determinisitic application of Grover's algorithm

Again for this algorithm, we can show that its average running time is $O(\sqrt{N/t})$. In the following we will prove this fact after preparing some tools.

First let us see that at some point of the for-iteration of the algorithm, the angle after executing $G(m)$ in this iteration gets reasonably close to $\pi/2$.

Claim. There exists some integer $i_0 > 0$ such that

$$\frac{\pi}{3} \;\le\; (2 \cdot 2^{i_0} + 1)\theta \;\le\; \frac{2\pi}{3}.$$

Furthermore, since $\theta \approx \sqrt{t/N}$, we have $2^{i_0} < (\pi/3)\sqrt{N/t}$.

Proof. Clearly, $(2 \cdot 2^i + 1)\theta$ $(\approx (2 \cdot 2^i + 1)\sqrt{t/N}) > \pi/3$ for some $i > 0$. Let i be the largest integer such that $(2 \cdot 2^i + 1)\theta < \pi/3$ holds. Then we have

$$(2 \cdot 2^{i+1} + 1)\theta \;<\; 2(2 \cdot 2^i + 1)\theta \;<\; \frac{2\pi}{3}.$$

Therefore, the claim holds with $i_0 = i + 1$.

Below we will keep using i_0 to denote the one satisfying this claim. Also let W denote the above range of angles; that is, $W = [\pi/3, 2\pi/3]$. As stated in the above claim, the angle at the i_0th for-iteration gets into W; then the probability that a solution is found at this for-iteration is, as we will see below, more than $1 - (1/4)^k$. In general, for any i, let us simply call the angle $(2 \cdot 2^i + 1)\theta$ *the angle at the ith for-iteration*. Recall that the success probability is defined by $\sin^2$ of angles; hence, we had better discuss angles with their residues under modulo π. Thus, in the following, by "an angle" we mean the residue of a real angle under modulo π. For any for-iteration, we have the following relation between the angle and the failure probability at this for-iteration.

Claim. For any $i > 0$, let ω be the angle at the ith for-iteration. Let $\alpha = |\pi/2 - \omega|$ and $\delta = \sin^2 \alpha$. Then the probability that no solution is found at this for-iteration is δ^k. (Note that if $\omega \in W$, then we have $\delta \le 1/4$.)

Proof. Since the angle at the ith for-iteration is ω, the probability that some solution is found by one execution of $G(m)$ (where $m = 2^i$) is

$$t \cdot \left(\frac{1}{\sqrt{t}} \sin \omega\right)^2 \;=\; \sin^2 \omega.$$

Hence, the probability that no solution is found at this iteration is

$$(1 - \sin^2 \omega)^k \;=\; (\cos^2 \omega)^k \;=\; \left(\sin^2 |\pi/2 - \omega|\right)^k \;=\; (\sin^2 \alpha)^k \;=\; \delta^k.$$

It would be nice if we could argue that the angle belongs to W reasonably often. Unfortunately, however, if the angle (at some for-iteration) gets very close to $\pi/2$, then it takes rather long time to have an angle in W again. Note, on the other hand, that if the angle is close to $\pi/2$, then the probability of finding a solution at this iteration is close to 1. The crucial point of our analysis is to estimate this trade-off.

For simplifying our discussion, instead of an angle ω, we will argue by using $\alpha = |\pi/2 - \omega|$, which we call a *co-angle*. For example, the above claim shows that if the co-angle (at some for-iteration) gets less than $\pi/6$, then the error

probability at this iteration becomes less than $(1/4)^k$. Let A denote this good co-angle range; that is, $A = \{\alpha : 0 \leq \alpha \leq \pi/6\}$.

For our analysis, we estimate the number of for-iterations until having a co-angle in A again.

Claim. Consider any ith for-iteration with the co-angle $\alpha \in A$. (We assume that $\alpha > 0$.) Define $h = \log\lceil \pi/(3\alpha)\rceil$. Then there exists some $\overline{h}$, $1 \leq \overline{h} \leq h$, such that the co-angle at the $(i + \overline{h})$th for-iteration gets into A again.

Remark. For simplifying our discussion. the argument here is a bit unprecise. More specifically, it may be the case that the co-angle at the $(i+\overline{h})$th for-iteration is slightly larger than $\pi/3$; but the difference is at most $+3\theta$, which should be small enough under our assumption on t/N.

Proof. Here we show that the angle at the $(i + \overline{h})$th for-iteration belongs to the range $W = [\pi/3, 2\pi/3]$.

Since the claim clearly holds if $\alpha = \pi/6$, we may assume that $\alpha < \pi/6$, which implies that $h \geq 2$. Recall that the definitions of angle and co-angle; that is, we have $(\omega =)\ \pi/2 - \alpha = (2 \cdot 2^i + 1)\theta$, where ω is the angle at the ith for-iteration. (Here we consider only the case that $\omega < \pi/2$. The case that $\omega > \pi/2$ can be treated in a similar way.) Then for any h', $2 \leq h' \leq h$, we can estimate the angle ω' at the $(i + h')$th for-iteration as follows.

$$\begin{aligned}
\omega' &= (2 \cdot 2^{i+h'} + 1)\theta \pmod{\pi} \\
&= 2^{h'} \cdot ((2 \cdot 2^i + 1)\theta) - (2^{h'} - 1)\theta \pmod{\pi} \\
&= 2^{h'} \cdot \left(\frac{\pi}{2} - \alpha\right) - (2^{h'} - 1)\theta \pmod{\pi} \\
&= -(2^{h'}\alpha + (2^{h'} - 1)\theta - m\pi) \pmod{\pi}.
\end{aligned}$$

Here m is some nonnegative integer such that $0 \leq 2^{h'}\alpha + (2^{h'} - 1)\theta - m\pi < \pi$. Since $\sin\omega' = \sin(\pi - \omega')$, we can discuss by using $\overline{\omega}' = 2^{h'}\alpha + (2^{h'} - 1)\theta - m\pi$ instead of the above ω'.

Now we consider the first $h' \leq h$ such that $2^{h'}\alpha + (2^{h'} - 1)\theta \geq \pi/3$. Such h' exists because we have $2^h\alpha \geq \pi/3$ from our choice of h. Furthermore, we may assume that $h' \geq 2$, because in the special case that $h' = 1$, we can easily show that $\overline{h} = 2$ satisfies the claim. Also notice that $\overline{\omega}' = 2^{h'}\alpha + (2^{h'} - 1)\theta$; that is, $m = 0$ for this h'.

Clearly, we are done if $\overline{\omega}' \leq 2\pi/3$. But if otherwise, i.e., if $2^{h'}\alpha + (2^{h'} - 1)\theta > 2\pi/3$, then the angle $\overline{\omega}'' = 2^{h'-1}\alpha + (2^{h'-1} - 1)\theta$ $(\approx (2^{h'}\alpha + (2^{h'} - 1)\theta)/2)$ should be $\geq \pi/3$. On the other hand, since $2^{h'}\alpha + (2^{h'} - 1)\theta < \pi$, we have $\overline{\omega}'' = 2^{h'-1}\alpha + (2^{h'-1} - 1)\theta < \pi/2$ $(< 2\pi/3)$. Therefore, the claim is satisfied with either $\overline{h} = h'$ or $\overline{h} = h' - 1$.

Next we divide the co-angle range A. Below let j be any nonnegative integer. Let $\delta_j = 2^{-(j+2)}$, and let $\widehat{\alpha}_j$ be the co-angle $< \pi/2$ such that $\delta_j = \sin^2\widehat{\alpha}_j$. (For example, we have $\delta_0 = 1/4$ and $\widehat{\alpha}_0 = \pi/6$.) Then define $A_j = \{\alpha : \widehat{\alpha}_{j+1} < \alpha \leq \widehat{\alpha}_j\}$. Also we define h_j as follows.

$$h_j \;=\; \left\lceil \log \frac{\pi}{3\widehat{\alpha}_{j+1}} \right\rceil .$$

Since $\widehat{\alpha}_j$ gets very small, we may assume that $\sin \widehat{\alpha}_j \approx \widehat{\alpha}_j$. In particular, we may assume that $\widehat{\alpha}_{j+1} \geq \widehat{\alpha}_j/2$. (In fact, we would have $\widehat{\alpha}_{j+1} \geq \widehat{\alpha}_j/\sqrt{2}$ for large j.) Hence, we have $h_{j+1} \leq h_j + 1$.

Here let us summarize our discussion. Consider any ith for-iteration, and suppose that the co-angle at this iteration belongs to A_j. Then it follows from the above claims that (i) the probability that the algorithm does not halt at this for-iteration is $\leq \delta_j^k$ $(= 2^{-(j+2)k})$, and (ii) the co-angle gets into A again before the $(i + h_j)$th for-iteration.

Now we are ready to prove our theorem.

Theorem 3. *The average number of G-steps executed in Deterministic Grover (the algorithm of Figure 2) is* $(8\pi/3)\sqrt{N/t}$.

Proof. Recall that the co-angle belongs to A at the i_0th for-iteration. We may assume that it is indeed the first for-iteration having a co-angle in A. In general, we use i_1, i_2, ..., to denote the first, the second, ... for-iteration (after the i_0th for-iteration) where the co-angle belongs to A. Also let α_0, α_1, ... denote the co-angles at the i_0th, i_1th, ... iterations.

To bound the expected number of executed G-steps, we may assume that the execution of the algorithm reaches to the i_0th for-iteration. Thus, we first estimate the number of G-steps executed by the end of the i_0th for-iteration, which is bounded as follows.

$$k \cdot (1 + 2 + 2^2 + \cdots + 2^{i_0}) \;=\; k \cdot (2^{i_0+1} - 1) \;<\; \frac{2k\pi}{3}\sqrt{\frac{N}{t}}. \tag{1}$$

Next we analyze the expected number E_0 of G-steps executed after the i_0th for-iteration. (Precisely speaking, E_0 that we will analyze below is the expected number E_0 of G-steps executed after the i_0th for-iteration under the condition that the i_0th for-iteration is executed.) Although we know that the co-angle α_0 at the i_0th for-iteration is in A, we do not know which A_j it belongs to. But for each $j \geq 0$, by assuming that $\alpha \in A_j$, we can estimate the failure probability and the length to the i_1th for-iteration. Thus, we can bound E_0 in the following way. (Here we use E_1 to denote the amount corresponding to E_0; that is, E_1 is the expected number of G-steps executed after the i_1th for-iteration under the condition that the i_1th for-iteration is executed.)

$$\begin{aligned} E_0 &\leq \sum_{j\geq 0} \Pr\{\text{ no solution is found at the } i_0\text{th for-iteration} \mid \alpha_0 \in A_j \} \\ &\qquad \times (\#\text{ of G-steps executed during } ((i_0+1) \sim i_1)\text{th for-iteration} + E_1) \\ &\leq \sum_{j\geq 0} \delta_j^k \cdot \left(k \cdot (2^{i_0+1} + 2^{i_0+2} + \cdots + 2^{i_0+h_j}) + E_1\right) \end{aligned}$$

$$= \sum_{j\geq 0} \delta_j^k \cdot k \cdot \left(2^{i_0+h_j+1} - 1\right) + \sum_{j\geq 0} \delta_j^k \cdot E_1$$

$$\leq k2^{i_0} \cdot \left(\sum_{j\geq 0} \delta_j^k \cdot 2^{h_j+1}\right) + \sum_{j\geq 0} \delta_j^k \cdot E_1.$$

Here by using the fact that $h_j \leq h_0 + j$ and $h_0 \leq 2$, and our choice of δ_j and k (i.e., $\delta_j = 2^{-j+2}$ and $k = 2$), we can bound the last expression as follows.

$$E_0 \leq k2^{i_0} \cdot \left(\sum_{j\geq 0} 2^{-(j+2)k} \cdot 2^{h_0+j+1}\right) + \sum_{j\geq 0} \delta_j^k \cdot E_1$$

$$= k2^{i_0} \sum_{j\geq 0} 2^{-j-1} + \sum_{j\geq 0} \delta_j^k \cdot E_1 \;=\; k2^{i_0} + \sum_{j\geq 0} \delta_j^k \cdot E_1. \tag{2}$$

At this point, let us see how E_1 is estimated. Notice that E_1 depends on the value of i_1. Otherwise, the calculation is quite similar to the one for E_0. (Here we use E_2 denoting the amount corresponding to E_1 for i_2.)

$$E_1 \leq \sum_{j\geq 0} \Pr\{\text{ no solution is found at the } i_1\text{th for-iteration} \mid \alpha_1 \in A_j\}$$

$$\times (\#\text{ of G-steps executed during } ((i_1+1) \sim i_2)\text{th for-iteration} + E_2)$$

$$\leq \sum_{j\geq 0} \delta_j^k \cdot \left(k \cdot (2^{i_1+1} + 2^{i_1+2} + \cdots + 2^{i_1+h_j}) + E_2\right)$$

$$\leq k2^{i_1} \cdot \left(\sum_{j\geq 0} \delta_j^k \cdot 2^{h_j+1}\right) + \sum_{j\geq 0} \delta_j^k \cdot E_2 \;\leq\; k2^{i_1} + \sum_{j\geq 0} \delta_j^k \cdot E_2. \tag{3}$$

We substitute E_1 of (2) by (3). Here notice that i_1 depends on the choice of α_0; in fact, if $\alpha_0 \in A_j$, then it is bounded by $i_0 + h_j$ ($\leq i_0 + h_0 + j \leq i_0 + j + 2$). Thus, we have the following bound.

$$E_0 \leq k2^{i_0} + \sum_{j_1\geq 0} \delta_{j_1}^k \cdot \left(k2^{i_1} + \sum_{j_2\geq 0} \delta_{j_2}^k \cdot E_2\right)$$

$$\leq k2^{i_0} + \sum_{j_1\geq 0} 2^{-2(j_1+2)} \cdot k2^{i_0+j_1+2} + \sum_{j_1\geq 0} \delta_{j_1}^k \cdot \left(\sum_{j_2\geq 0} \delta_{j_2}^k \cdot E_2\right)$$

$$= k2^{i_0} + k2^{i_0} \sum_{j_1\geq 0} 2^{-(j_1+2)} + \sum_{j_1\geq 0} \delta_{j_1}^k \cdot \left(\sum_{j_2\geq 0} \delta_{j_2}^k \cdot E_2\right)$$

$$= k2^{i_0} + \frac{1}{2} \cdot k2^{i_0} + \sum_{j_1\geq 0} \delta_{j_1}^k \cdot \left(\sum_{j_2\geq 0} \delta_{j_2}^k \cdot E_2\right).$$

We can similarly expand E_2, E_3, ..., thereby deriving the following bound.

$$E_0 \;\leq\; k2^{i_0} + 2^{-1} \cdot k2^{i_0} + 2^{-2} \cdot k2^{i_0} + \cdots \;=\; 2k2^{i_0} \;\leq\; \frac{2k\pi}{3}\sqrt{\frac{N}{t}}. \qquad (4)$$

Now the bound of the theorem is immediate from (1) and (4), and our choice of k.

References

[BV97] E. Bernstein and U. Vazirani, Quantum complexity theory, *SIAM Journal on Computing* 26, 1411–1473 (1997).

[BBHT96] M. Boyer, G. Brassard, P. Hoyer, and A. Tapp, Tight bounds on quantum searching, quant-ph/9605034.

[For00] L. Fortnow, One complexity theorist's view of quantum computing, quant-ph/0003035.

[Gro96] L.K. Grover, A fastquantum mechanical algorithm for database search, in *Proc. 28th Annual ACM Symposium on Theory of Computing*, ACM, 212–219 (1996).

[Gro97] L.K. Grover, Quantum mechanics helps in searching for a needle in a haystack, *Phys.Rev.Lett* 79, 325–328 (1997).

[Gru99] J. Gruska, *Quantum Computing*, McGrow-Hill, 1999.

[Hos99] A. Hosoya, *Lectures on Quantum Computation* (in Japanese), Science Pub. Co., 1999.

Random Instance Generation for MAX 3SAT (Extended Abstract)

Mitsuo Motoki

Tokyo Women's Medical University
Kawadacho, Sinjuku, Tokyo, 162-8662
mmotoki@ior.twmu.ac.jp

1 Introduction

MAX SAT is one of famous combinatorial optimization problems stated as follows: given a multiset of clauses, find an assignment that maximizes the number of satisfied clauses (that is equivalent to finding an assignment that minimizes the number of unsatisfied clauses). MAX 3SAT is restricted version of MAX SAT, that is, its input is restricted to a multiset of 3-clauses, i.e., each clause contains exactly 3 literals whose underlying variables are distinct each other. Since these problems are not only NP-hard problem, but MAX SNP-complete problem, there is no polynomial time approximation algorithm whose approximation ratio is close to 1 unless P = NP. Furthermore, Håstad showed that it is NP-hard to approximate MAX 3SAT within $8/7 - \varepsilon$ for any $\varepsilon > 0$ [5]. In spite of these negative results, many polynomial time approximation algorithms with proven approximation ratio for MAX SAT have been proposed [3,4,7].

To evaluate these approximation and exact algorithms experimentally, test instances are necessary. While there are some benchmark problems for MAX SAT, we still do not have enough number of problems, and we would like to have sure way generating nontrivial test instance systematically. In particular, it would be nice if we have an efficient test instance generators for MAX SAT.

Test instance generator have been studied for NP problems (see, e.g. [1, 2, 6, 8]). On the other hand, for optimization problems like MAX SAT, no generator, as far as the author knows, has been proposed in the literature.

When one evaluates an approximation algorithm, test instances should have their optimal solution otherwise we can not determine how good approximated solution are obtained. In spite of necessity of optimal solution, it seems hard to check whether given a (candidate of) optimal solution is really optimal. In fact, in the case of MAX SAT, it is co-NP complete to check the optimality of a given solution.

Since the hardness of generating a test instance with its optimal solution seems unavoidable, we consider two alternative approaches. The first one is to estimate (analytically and experimentally) the expected number $E[s(F)]$ of optimally satisfied clauses of F, when MAX SAT instance F is generated under a certain distribution. Suppose that we can estimate $E[s(F)]$ with reasonable precision. Then even if we can not supply optimal solutions, we could still test

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 502–508, 2001.
© Springer-Verlag Berlin Heidelberg 2001

the average performance of a given MAX SAT algorithm. That is to say, we just run it on generated instances and compare the average number of clauses satisfied by obtained solution with $E[s(F)]$. The second approach is to design a test instance generator that provides both an instance and its candidate of optimal solution that is really optimal with high probability.

In this paper, we obtain some basic results for these two approaches. For the first one, we analyze the minimum number $u(F)$ of unsatisfiable clauses of given formula F. We obtain two lower bounds on its expectation $E[u(F)]$, $\Omega(m/\ln m)$ and $m/8 - O(\sqrt{mn})$, for random 3CNF formula with m clauses over n variables. We also analyze the expectation experimentally. For the second approach, on the other hand, we propose a simple random instance generator that, for given u, generates an instance and an assignment (i.e., a candidate for the optimal solution) that leaves just u clauses unsatisfied. Furthermore, we prove that the generated solution is indeed optimal with high probability if the number of clauses is $\Omega(n \ln n) + \Omega(n\sqrt{u \ln n})$.

Hereafter, we describe basic notations. Let $V = \{v_1, \ldots, v_n\}$ be propositional variables. We represent true and false by 1 and 0 respectively. An assignment t is a mapping from variables to Boolean values, $v_1, \ldots, v_n \to \{0,1\}^n$. We denote by T_d a set of assignments whose Hamming distance from 1^n is d. A *literal* is a propositional variable v or its *negation* $\overline{v}$. We use a complement-free set of exactly 3 literals to denote a *3-clause*, a disjunction of exactly 3 literals. Let *3CNF* formula F be a multiset of 3-clauses, and we denote $|F|$ by m. We say F is a *random 3CNF formula* with m clauses over n variables if each clause in F is chosen with replacement from possible $8\binom{n}{3}$ clauses over n variables uniformly at random. For any F and t, we denote the number of unsatisfied clauses in F at an assignment t by $u(F,t)$. Furthermore, $u(F)$ is the minimum number of unsatisfied clauses in F for all assignments, that is, $u(F) = \min_{t \in \{0,1\}^n} u(F,t)$. We say $u(F)$ is the *optimal value* of F.

In this abstract, we omitted proofs and some explanations. See research report in http://www.is.titech.ac.jp/.

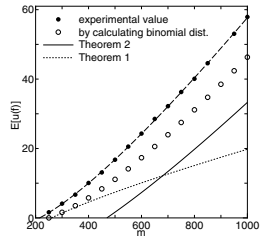
2 The Average Optimal Value for Random 3CNF Formula

We consider the expected number $E[u(F)]$ of unsatisfiable clauses on the optimal solution. There exists an obvious upper bound $m/8$ since it is the upper bound of $u(F)$. Here we prove two lower bounds of $E[u(F)]$.

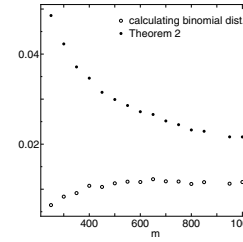
Theorem 1. *For any $m \geq n \ln 2/\ln(8/7) \approx 5.19n$, the expected number of unsatisfiable clauses on optimal assignment satisfies the following bound,*

$$E[u(F)] \geq \frac{m \ln 8/7 - n \ln 2 + \ln(m/7 - 1)}{\ln m/7}(1 - 7/m).$$

The sketch of the proof is as follows. For any assignment t, the probability $\Pr[u(F,t) = v]$ that the number of unsatisfiable clauses in F on t is exactly v is estimated as $\binom{m}{v}\left\{\frac{1}{8}\right\}^v\left\{\frac{7}{8}\right\}^{m-v}$. By using this estimation, we can bound



(a) the average of $u(F)$



(b) the ratio to m of difference between the experimental and the theoretical value

Fig. 1. Experimental results

the probability that the number of unsatisfiable clauses in F on its optimal assignment is larger than v. Finally, we use Markov's inequality.

In the Theorem 1, rounding $\binom{m}{v}$ by m^v causes the denominator of upper bound, $\ln m/7$. Thus we need to estimate the probability that $u(F)$ is larger than v more tightly. The probability $\Pr[u(F) \geq v]$ includes partial sum of binomial distribution. For sufficiently large m, we can approximate binomial distribution to normal distribution by using central limit theorem. This observation leads us to the next result.

Theorem 2. *For any* $m \geq 7n \ln 2 + \sqrt{7n \ln 2 - 16}\sqrt{7n \ln 2} - 8 \approx 9.7n - 8$, *the expected number* $E[u(F)]$ *of optimally unsatisfiable clauses satisfies following bound,*

$$E[u(F)] \geq \frac{1}{8}\left(m - \sqrt{14mn \ln 2} + 8\right)\left(1 - \frac{4}{\sqrt{7\pi nm \ln 2}}\right).$$

We compare theoretical lower bounds with the average number of unsatisfiable clauses on optimal assignment that we obtained through some experiments.

Now we describe the setup used in the experiments. We fixed the number of variables to 50, and we selected the numbers of clauses m from 250 to 1000 at intervals of 50 (we skipped 900). For each m, we checked the average optimal value to 100 (50, for over 600 clauses) random 3CNF.

We show results in Fig. 1(a) , along the horizontal axis we give the number of clauses m, and along the vertical axis we give the average number of unsatisfiable clauses on optimal. Big black dots are results of experiments, and dotted line and solid line are lower bounds of Theorem 1 and Theorem 2 resp. Furthermore, white circle are values obtained by calculating the partial sum of binomial distribution, $\arg\max_v \left(v \left(1 - \sum_{j=0}^{v} \binom{m}{v}(1/8)^v(7/8)^{m-v}\right)\right)$. With this graph, it seems that behavior of experimental results is similar to that of lower bound of Theorem 2.

In Fig. 1(b), we show the ratio of difference between the experimental and the theoretical value to m. Black (white, resp.) dots are the ratio of the difference between the experimental value and the lower bound of Theorem 2 (the value that obtained by calculating the binomial distribution, resp.). We can see that the difference is as much as $0.02m$. Thus we assume that the relation of $E[u(F)]$

with m forms $(1/8-c_1)m-\sqrt{c_2 m}+c_3$, and get $(1/8-0.00620)m-\sqrt{7.13m}+10.9$ (dashed line in Fig. 1(a)) by the least squares.

3 Random Instance Generation for MAX 3SAT

Our final goal is to generate MAX SAT instance with its optimal solution at random. When evaluating MAX SAT solver experimentally, the optimal solution is necessary since it is hard to determine whether obtained solutions are correct. On the contrary, it is also hard to check a (candidate of) optimal solution generated with an instance. Hence, we relax the condition of instance generator such that the candidate of optimal solution is correct with high probability, that is, for given n and u, our generator generates appropriate size MAX SAT instance over n variables and an assignment (for candidate of optimal solution) that unsatisfies exactly u clauses.

Our basic idea is as follows. Let F and $\hat{t}$ be the output instance and assignment of our instance generator for given n and u. We assume that 1^n is chosen as $\hat{t}$ w.l.o.g. Of course, F shall have exactly u clauses that are not satisfied by $\hat{t}$. These clauses shall be the form of $\{\overline{v_i}, \overline{v_j}, \overline{v_k}\}$. There exist some assignments in T_1 that satisfy some of these u clauses while the number of unsatisfied clauses by 1^n is exactly u. Therefore, for each of these u clauses, we add three more clauses forms of $\{v_i, \overline{v_{i_1}}, \overline{v_{i_2}}\}$, $\{v_j, \overline{v_{j_1}}, \overline{v_{j_2}}\}$ and $\{v_k, \overline{v_{k_1}}, \overline{v_{k_2}}\}$, respectively. Now each assignment in T_1 has exactly u unsatisfied clauses. Then we add appropriate number of clauses that satisfied by $\hat{t}$. Intuitively, $\hat{t}$ becomes one of optimal solutions of F with high probability when enough large number of clauses are added.

Now we introduce our instance generation algorithm formally. First, we define some sets of clauses C_{9-}, C_{9+}, C_5^i, C_{3-} and C_{3+} as follows:

$$
\begin{aligned}
C_{9-} &= \{\{\overline{v_{i1}}, \overline{v_{i2}}, \overline{v_{i3}}\}, \{v_{i1}, \overline{v_{i4}}, \overline{v_{i5}}\}, \{v_{i2}, \overline{v_{i6}}, \overline{v_{i7}}\}, \{v_{i1}, \overline{v_{i8}}, \overline{v_{i9}}\}\},\\
C_{9+} &= \{\{v_{i1}, v_{i2}, v_{i3}\}, \{\overline{v_{i1}}, v_{i4}, v_{i5}\}, \{\overline{v_{i2}}, v_{i6}, v_{i7}\}, \{\overline{v_{i1}}, v_{i8}, v_{i9}\}\},\\
C_5^i &= \{\{v_i, \overline{v_{i1}}, \overline{v_{i2}}\}, \{\overline{v_i}, v_{i3}, v_{i4}\}\},\\
C_{3-} &= \{v_{i1}, \overline{v_{i2}}, \overline{v_{i3}}\}, \text{and } C_{3+} = \{\overline{v_{i1}}, v_{i2}, v_{i3}\}.
\end{aligned}
$$

Hereby we show a pseudo code of our algorithm in Fig. 2. In this algorithm, u many sets in the form of C_{9-} (C_{9+}, resp.) guarantee that an assignment 1^n (0^n, resp.) does not satisfy u clauses exactly and each assignment in T_1 (T_{n-1}, resp.) does not satisfy at least u clauses. On the other hand, there will exist some assignments that does not satisfy less than u clauses. In such case, $\hat{t}$ is no longer the optimal solution of F. To avoid these assignments, we add r clauses satisfied by both of 1^n and 0^n (C_{3-} and C_{3+}). We also add b many sets of $\cup_{i=1}^n C_5^i$ to balance the number of occurrence of each literal. Intuitively, the larger r makes the higher probability that $\hat{t}$ is the optimal solution of F. Hence, we analyze the condition of r such that $\hat{t}$ is the optimal solution of generated instance.

Let random variable X_t be the number of clause sets forms of C_{9-}, C_{9+}, C_5^i, C_{3-}, and C_{3+} that contain at least one clause not satisfied t. It is clear that

input: the number of variables n, the desirable optimal value u,
the number of clause sets $\cup_{i=1}^{n} C_5^i$, b, the number of random clauses r.
begin
let $F = \emptyset$;
repeat u times **do**
choose 9 distinct variables $v_{r1}, \ldots, v_{r9}$ and add $C_{9-}(v_{r1}, \ldots, v_{r9})$ to F;
choose 9 distinct variables $v_{r1}, \ldots, v_{r9}$ and add $C_{9+}(v_{r1}, \ldots, v_{r9})$ to F;
repeat b times **do**
for $i = 1$ to n **do**
choose 4 distinct variables $v_{r1}, \ldots, v_{r4} \in V \setminus \{v_i\}$ and add $C_5^i(v_{r1}, \ldots, v_{r4})$ to F;
repeat r times **do**
choose 3 distinct variables v_{r1}, v_{r2}, v_{r3} and add $C_{3-}(v_{r1}, v_{r2}, v_{r3})$ to F;
choose 3 distinct variables v_{r1}, v_{r2}, v_{r3} and add $C_{3+}(v_{r1}, v_{r2}, v_{r3})$ to F;
choose a truth assignment $\hat{t} \in \{0,1\}^n$ at random;
flip literals whose under lying variables are assigned 0 on $\hat{t}$;
output F and $\hat{t}$;
end.

Fig. 2. Algorithm.

$u(F,t)$ is at least u for any t in T_0 $(= \{1^n\})$, T_1, T_{n-1} and T_n $(= \{0^n\})$. Thus our goal is to show the condition that X_t is at least u for any t in T_d $(2 \leq d \leq n-2)$ with high probability (we remark that X_t is less than $u(F,t)$).

First we consider μ_d the expectation of X_t on arbitrary assignments t in T_d.

Lemma 1. *For any d $(0 \leq d \leq n)$, for any assignment $t \in T_d$, let μ_d be the expected number of clause sets C_{9-}, C_{9+}, C_5, C_{3-} and C_{3+} in F such that each set contains at least one clause not satisfied by t (the expectation of X_t). Then*

$$\mu_d \geq u - u\frac{d(n-d)}{n(n-1)} + b\frac{d(n-d)}{n-1} + r\frac{d(n-d)}{n(n-1)}.$$

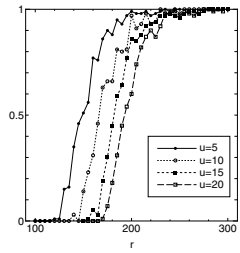
By using this lemma, we can use Chernoff bound to bound the probability that X_t is less than u and prove following theorem.

Theorem 3. *For any positive $\varepsilon < 1$, the probability that $\hat{t}$ is optimal solution of F is at least $1 - \varepsilon$ if*

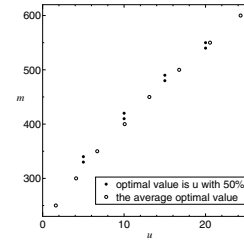
$$r \geq u - bn + 4(n-1)\ln 2n - 2(n-1)\ln \varepsilon + (n-1)\sqrt{2\ln 2n - \ln \varepsilon}\sqrt{2u}.$$

In theoretical analysis, we prove that our instance generator outputs instance with optimal value u with high probability if r is $u + \mathcal{O}(n \ln n) + \mathcal{O}(n\sqrt{u \ln n})$. But is such a large number really required? In fact, this upper bound becomes over 1300 even if $u = 5$, $n = 50$ and $\varepsilon = 1$. Thus we compare this theoretical bound with r obtained through experiments. Each experiment, we calculate the ratio of instances whose optimal value is u for 100 instances.

First, we estimate whether our algorithm can cover the average optimal value. We fix the number of variables n to 50 and b to 0. We select u from 5, 10, 15 and

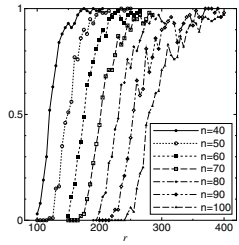


(a) the ratio of instances changing optimal value u

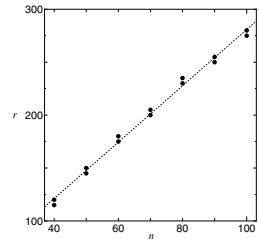


(b) the number of clauses and optimal value u (ratio is 0.5)

Fig. 3. Changing the value to be optimal u ($n = 50$, $b = 0$)



(a) the ratio of instances whose optimal value is 5



(b) r for each n where 50% instances have optimal value 5

Fig. 4. Changing the number of variables n ($u = 5$, $b = 0$)

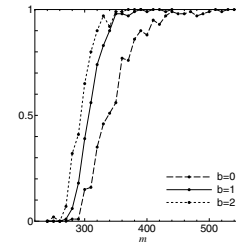


Fig. 5. Changing b. ($n = 50$, $u = 5$)

20. Results are in Fig. 3(a). The intervals of results are smaller as opposed to u. For each u, we pick up two values of $m = 8u + 2r$, the number of total clauses, such that one is the largest m where the ratio is less than 50% and the other is the smallest m where the ratio is larger than 50%. We compare this result with the average optimal value obtained experimentally in Section 2 (Fig. 3(b)). From the figure, we can see that the number of clauses that the ratio of instance with optimal value u is just 50% is near the the number of clauses whose average optimal value is u. Thus, we consider that our algorithm generates an instance whose optimal solution is given with high probability if the optimal value is quite smaller than the average optimal value.

Second, we want to estimate whether the main term of upper bound on r is $n \ln n$ or not. We choose n from 40 to 100 at intervals of 10. And we fix u to 5 and b to 0. Results are in Fig. 4(a). Then we also pickup two values of r where the ratio of instances with given optimal solution is just smaller and larger than 0.5 for each n. In Fig. 4(b), the vertical axis is value of r where the ratio of instances with given optimal value is just stepping over 0.5, the horizontal axis is the number of variables n. From this result, the relation of r and n seems linear or the constant of $n \ln n$ is very small. Hence, we try to fit by linear function, and get $r = 2.6n + 14.8$ (dotted line).

Finally, we change b, the number of clause sets $\cup_{i=1}^{n} C_5^i$. In the theoretical upper bound, there exists the term $-bn$. This means that adding b sets of $\cup_{i=1}^{n} C_5^i$ is equivalent to adding bn random clauses of the each form C_{3-} and C_{3+} ($2bn$ clauses). We fix the number of variables n to 50 and the optimal value u to 5 again. We choose b from 1, 2 and 3. Results are in Fig. 5, the vertical axis is the ratio of instances whose optimal value is 5 and the horizontal axis is the total number of clauses m. We can easily see that b diminishes the total number of clauses. Furthermore, it contribute converging the ratio to 1 much faster.

References

1. D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. *Proc. of AAAI-2000/IAAI-2000*, 2000.
2. Y. Asahiro, K. Iwama, and E. Miyano. Random generation of test instances with controlled attributes. Johnson and Trick, eds., *Cliques, Coloring, and Satisfiability*, DIMACS vol. 26, pp. 377–393. AMS, 1996.
3. M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM J. Discrete Math.*, 7:656–666, 1994.
4. M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.
5. J. Håstad. Some optimal inapproximability results. *Proc. of STOC'97*, pp. 1–10, 1997.
6. S. Horie and O. Watanabe. Hard instance generation for SAT (extended abstract). *Proc. of ISAAC'97*, LNCS 1350, pp. 22–31, 1997.
7. H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? *Proc. of FOCS'97*, pp. 406–415, 1997.
8. M. Motoki and R. Uehara. Unique solution instance generation for the 3-Satisfiability (3SAT) problem. Gent, van Maaren, and Walsh, eds., *SAT2000 Highlights of Satisfiability Research in the Year 2000*, pp. 293–307. IOS Press, 2000.

The Euclidean Bottleneck Steiner Tree and Steiner Tree with Minimum Number of Steiner Points

Dingzhu Du[1], Lusheng Wang[2], and Baogang Xu[3]

[1] Department of Computer Science and Engineering,
University of Minnesota, Minnesota, MN 55455, USA.
dzd@cs.umn.edu
[2] Department of Computer Science,
City University of Hong Kong, Kowloon, Hong Kong.
lwang@cs.cityu.edu.hk
[3] Institute of Systems Sciences, Academy of Math. & Systems Sciences,
Chinese Academy of Sciences, Zhongguancun, Beijing, 100080.
bgxu@staff.iss.ac.cn

Abstract. We study variations of *Steiner tree* problem. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n terminals in the Euclidean plane. For a positive integer k, the *bottleneck Steiner tree problem* ($BSTP$ for short) is to find a Steiner tree with at most k Steiner points such that the length of the longest edges in the tree is minimized. For a positive constant R, the *Steiner tree problem with minimum number of Steiner points* ($STP - MSP$ for short) asks for a Steiner tree such that each edge in the tree has length at most R and the number of Steiner points is minimized. In this paper, we give (1) a ratio-$\sqrt{3} + \epsilon$ approximation algorithm for BSTP, where ϵ is an arbitrary positive number; (2) a ratio-3 approximation algorithm for STP-MSP with running time $O(n^3)$; (3) a ratio-$\frac{5}{2}$ approximation algorithm for STP-MSP.

1 Introduction

Given a set P of terminals in the Euclidean plane, a *Steiner tree* on P is an acyclic network interconnecting the terminals. Every vertex in a Steiner tree other than terminals is called a *Steiner point*.

A variation of Steiner tree problem, named *bottleneck Steiner tree problem* (BSTP for short) is defined as follows: given a set P of n terminals and a positive integer k, we want to find a Steiner tree with at most k Steiner points such that the length of the longest edges in the tree is minimized. Contrary to the classic Steiner tree problem, degree-2 Steiner points are allowed in BSTP. Instead of minimize the total length of the tree, here we want to minimize the length of the longest edges. The problem has applications in the design of wireless communication networks [8]. Due to budget limit, we could put totally $n + k$ stations in the plane, n of them must be located at given points. We would like to have the distances between stations as small as possible. The problem here is

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 509–518, 2001.
© Springer-Verlag Berlin Heidelberg 2001

to choose the locations for the other k stations to minimize the longest distance between stations. For other related variations and applications, see [1,2,4,7].

The problem is NP-hard. In [8], it shows that the problem cannot be approximated in polynomial time within performance ratios 2 and $\sqrt{2}$ in the rectilinear plane and the Euclidean plane, respectively. Moreover, a ratio-2 approximation algorithm was given for both the rectilinear plane and the Euclidean plane [8]. For the rectilinear plane, the performance ratio is the best possible. A ratio-1.866 approximation algorithm was given for the Euclidean plane [9]. In this paper, we give a randomized approximation algorithm with performance ratio $\sqrt{3}+\epsilon$ for the Euclidean plane, where ϵ is an arbitrary positive number.

Another variation of Steiner tree problem considered here is the *Steiner tree problem with minimum number of Steiner points*, denoted by $STP-MSP$ for short, is defined as follow: Given a set P of n terminals in the Euclidean plane, and a positive constant R, STP-MSP asks for a tree T spanning a superset of P such that each edge in the tree has length at most R and the number $C(T)$ of Steiner points is minimized. In the STP-MSP problem, Steiner points of degree 2 are possible. The STP-MSP problem has an important application in wavelength-division mnltiplexing optical network design[3,5]. In [4], Lin and Xue proved that the STP-MSP problem is NP-hard and gave a ratio-5 approximation algorithm. In [1], Chen, Du *et al.* showed that the algorithm given by Lin and Xue [4] has performance ratio exactly 4, and they also presented a new $O(n^4)$-time approximation with performance ratio at most 3, where n is the number of given terminals. In this paper, we give a $O(n^3)$-time approximation with performance ratio at most 3, and a randomized approximation with performance ratio at most $\frac{5}{2}$.

A *full component* of a Steiner tree is a subtree in which each terminal is a leaf and each internal node is a Steiner point. A Steiner tree for n terminals is a *k-restricted Steiner tree* if each full component spans at most k terminals.

2 Ratio-$\sqrt{3}$ Approximation Algorithm for BSTP

In this section, we give a ratio-$\sqrt{3}$ approximation algorithm for BSTP in the Euclidean plane.

Theorem 1 *Let T be an optimum Steiner tree for BSTP. Then, there is a 3-restricted Steiner tree with the same number of Steiner points as T such that the longest edge in the tree is at most $\sqrt{3}$ times the optimum.*

Proof: We assume that T is rooted by arbitrarily selecting a Steiner point as its root. We will modify T bottom up into a 3-restricted Steiner as described in the theorem. Without loss of generality, we assume that T is a full Steiner tree.

We organize the nodes in T level by level (ignoring degree-2 Steiner points). Level 1 is the lowest level. Level i is the level above level $i-1$. Let v be a node at level 3, v' a child of v. If v' is a Steiner point, we can assume that the degree of v' is 3. Otherwise, we can always find two children of v', say a and b, such that $\angle av'b \leq 120^o$. Without loss of generality, assume $|av'| \geq |bv'|$, and assume

the path P from a to v' has m degree-2 Steiner points. Then we construct a new Steiner tree T' by removing all the edges on P, and directly connecting a and b with m degree-2 Steiner points so that the length of each edge in the segment ab is at most $\sqrt{3}$. Now, we need only to consider the tree obtained from T' by removing a and all the degree-2 nodes on the path connecting a and b in T'.

In the rest, we assume that the degree of v' is at most 3. We consider two cases.

Case 1. Every edge below v in T has length no more than 1. We consider the case where v has 4 grandchildren, and leave the case where v has 3-children to the interested readers.

We first consider that v is a degree-3 node. (See Figure 1: (a).) In this case, we can assume that $\angle bv'c > 120^\circ$ and $\angle dfe > 120^\circ$. Therefore, we have $\min\{\angle dfv, \angle efv\} < 120^\circ$ and $\min\{\angle cv'v, \angle bv'v\} < 120^\circ$, i.e.,

$$\min\{|dv|, |ev|\} < \sqrt{3} \quad \text{and} \quad \min\{|cv|, |bv|\} < \sqrt{3}. \tag{1}$$

Without loss of generality, assume that

$$\angle dfv = \theta = \min\{\angle vv'v, \angle vv'c, \angle dfv, \angle efv\}, \quad \text{and} \quad \angle vv'c \leq \angle vv'b. \tag{2}$$

We will find a point h on edge vv' such that $\max\{|ch|, |bh|, |dh|\} \leq \sqrt{3}$, and construct a new tree by removing nodes v' and f, adding edges ch, cv, dh, bh, and connecting d and e directly with a Steiner point w on the middle of de. (See Figure 1:(b).) Then, we can continue the modification process with $n-3$ terminals in $P \cup \{v\} - \{b, c, d, e\}$.

By (1) and (2), we know that $|ch| \leq \sqrt{3}$ for any h on the edge vv'. So, we need only to choose an h to guarantee that $|bh| \leq \sqrt{3}$ and $|dh| \leq \sqrt{3}$.

First we suppose that $\theta < 90^\circ$, then one can easily check that the point h on edge vv' with $|vh| = 2 - \sqrt{3}$ satisfies the requirement.

Now we suppose that $90^\circ \leq \theta < 120^\circ$. By (2), we know that $\angle bv'v < 360^\circ - 120^\circ - \theta = 240^\circ - \theta$. We choose h to be the point on edge vv' such that $|vh| =$

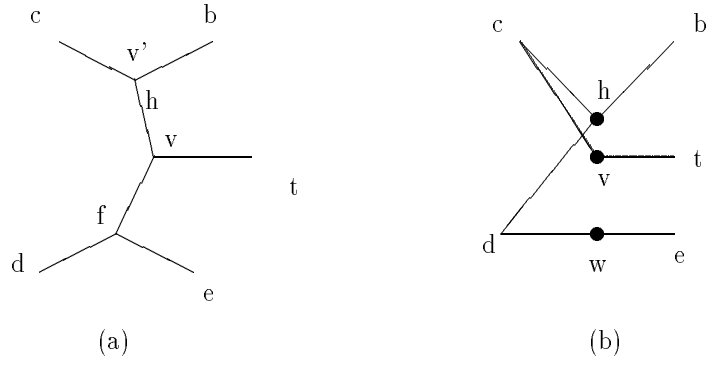


Fig. 1. (a) The original tree. (b) The modified tree.

$\sqrt{3} - |dv|$. By (1), $|vh| > 0$. It is eay to see that

$$|v'h| \leq 1 - \sqrt{3} + \sqrt{2 - 2\cos\theta}. \tag{3}$$

By triangle inequality,

$$|dh| \leq |dv| + |vh| = \sqrt{3}. \tag{4}$$

Using (3), we have

$$\begin{aligned} |bh|^2 &= |bv'|^2 + |v'h|^2 - 2|bv'||v'h|\cos\angle bv'v \\ &\leq 1 + (1 - \sqrt{3} + 2\sin\frac{\theta}{2})^2 + 2(1 - \sqrt{3} + 2\sin\frac{\theta}{2})\sin(\theta + 30^\circ) = G(\theta) \end{aligned}$$

Set $H(\theta) = G(\theta) - (\sqrt{3})^2 = -2 + (1 - \sqrt{3} + 2\sin\frac{\theta}{2})^2 + 2(1 - \sqrt{3} + 2\sin\frac{\theta}{2})\sin(\theta + 30^\circ)$.

One can verify directly that $\frac{d^4H}{d\theta^4}(\theta) > 0$ for $90^\circ \leq \theta \leq 120^\circ$. This together with the fact that $\frac{d^3H}{d\theta^3}(90^\circ) < 0$ and $\frac{d^3H}{d\theta^3}(120^\circ) > 0$ indicates that $\frac{d^2H}{d\theta^2}(\theta) \leq \max\{\frac{d^2H}{d\theta^2}(90^\circ), \frac{d^2H}{d\theta^2}(120^\circ)\} < 0$ for $\theta \in [90^\circ, 120^\circ]$. So, $\frac{dH}{d\theta}(\theta)$ is strictly decreasing on $[90^\circ, 120^\circ]$, and then we have for $\theta \in [90^\circ, 120^\circ]$, $\frac{dH}{d\theta}(\theta) \geq \frac{dH}{d\theta}(120^\circ) > 0$. This means that $H(\theta)$ is strictly increasing on $[90^\circ, 120^\circ]$. So, $H(\theta) \leq H(120^\circ) = 0$. Combine this with (4), h is certainly a correct choice.

Case 2. Some edges below v have length greater than 1. Let u be a steiner point which is a child of v and has degree 3, x and y the two terminals connected to u.

Without loss of generality, suppose that $|ux| \leq |uy|$, ux and uy have l and k Steiner points(both including u), respectively. Let z be the point on uy such that $|uz| = l$. then, we can assume that zy contains $k - l$ Steiner points(including z), and ux and uz contain totally $2(l-1)$ Steiner points(not including u and z). We directly connect x and z and equally insert $\lceil 1.155l \rceil - 1$ Steiner points into xz. Then, the length of each edge on xz is

$$\frac{2l}{\lceil 1.115l \rceil} \leq \sqrt{3}.$$

After that, we have still $2(l-1) - (\lceil 1.115l \rceil - 1) = 2l - 1 - \lceil 1.115l \rceil = \lfloor 0.885l \rfloor - 1$ Steiner points which can be used to equally break ux into small edges. Then, by inserting $\lfloor 0.885l \rfloor - 1$ Steiner points into ux, each edge on ux has length at most

$$\frac{l}{\lfloor 0.885l \rfloor} \leq \sqrt{3} \quad \text{if } l \geq 3.$$

By this operation, u is changed into a vertex of degree 2 in the new tree, then we can continue the process with $n - 1$ terminals in $P \cup \{u\} \setminus \{x, y\}$.

The case when $l \leq 2$ can be proved by using almost the same argument as above. We leave it to the interested readers. ∎

Our algorithm for finding the approximation of an optimal 3-restricted Steiner tree is just the same as that of [9]. It is based on the following theorem about the minimum spanning tree for a weighted 3-hypergraph. A *hypergraph* $H = (V, F)$ consists of a set V(called vertices) and a set F(called edges) which is an arbitrary family of subsets of V. A *weighted hypergraph* $H = (V, F, w)$ is a hypergraph such that each edge e in F has a weight $w(e)$. An *r-hypergraph* $H_r(V, F, w)$ is a weighted hypergraph, each edge has cardinality at most r.

Theorem 2 *[6] There exists a randomized algorithm for the minimum spanning tree problem in 3-hypergraphs running in $poly(n, w_{max})$ time with probability at least 0.5, where n is the number of nodes in the hypergraph and w_{max} is the largest weight of edges in the hypergraph.*

We construct a weighted 3-hypergraph $H_3(V, F, w)$ from the set P of terminals. Here $V = P$, and $F = \{(a, b)|a \in P \text{ and } b \in P\} \cup \{(a, b, c)|a \in P \text{ and } b \in P \text{ and } c \in P\}$. To obtain the weight of each edge in F, we need to know B, the length of the longest edge in an optimal solution for BSTP. It is hard to find the exact value of B. However, we can find a B' that is at most $(1 + \epsilon)B$ for any ϵ in time $poly(n, \epsilon)$(interested reader can find the detail in [9]).

Theorem 3 *For any given ϵ, there exists a randomized algorithm that computes a Steiner tree with n terminals and k Steiner points such that the longest edge in the tree is at most $\sqrt{3} + \epsilon$ times of the optimum running in $\frac{1}{\epsilon} \times poly(n, k)$ time with probability at least 0.5.*

3 A Ratio-3 Approximation Algorithm for STP-MSP

In this section, we consider the STP-MSP problem. Given a set P of n terminals in the Euclidean plane, and a positive constant R, we want to find a Steiner tree with minimum number of Steiner points such that each edge in the tree has length at most R. In [1], Chen *et al.* presented an $O(n^4)$-time approximation with performance ratio at most 3. With a slightly modification, we may reduce the running time to $O(n^3)$.

Our algorithm is given in Figure 2.

Since we construct 3-stars in Step 2, the algorithm runs in $O(n^3)$ time.

For a given set P of terminals, a *minimum spanning tree* is a tree interconnecting the terminals in P with edge between terminals. For a given constant R, a *steinerized minimum spanning tree* is a tree obtained from a minimum spanning tree by inserting $\lceil \frac{|ab|}{R} \rceil - 1$ Steiner points to break each edge ab into small pieces of length at most R.

Let T be a Steiner tree and e be a line segment. $C(T)$ and $C(e)$ denote the numbers of Steiner points in T and e, respectively. $|e|$ denotes the length of e.

Lemma 4 *[1] Every steinerized minimum spanning tree has the minimum number of Steiner points among steinerized spanning trees.*

Algorithm A: 3-approximation Algorithm for STP-MSP

Input A set P of n terminals, a positive constant R.

Output A Steiner tree T_A in which each edge has length at most R.

0. Sort all $\frac{n(n-1)}{2}$ possible edges between the n terminals of P in length increasing order $e_1, e_2, \ldots, e_{\frac{n(n_1)}{2}}$, and set $T_A = \emptyset$;
1. **for** every e_i such that $|e_i| \leq R$ **do**
   if e_i connects two different connected components of T_A
   then put e_i into T_A;
2. **for** each subset of three terminals a, b, c respectively in three connected components of T_A **do**
   if there exists a points s within distance R from a, b and c
   then put the 3-star, consisting of three edges sa, sb, sc into T_A;
3. **for** every e_i **do**
   if e_i connects two different connected components of T_A
   then put e_i into T_A.

Fig. 2. The ratio-3 algorithms.

Lemma 5 *[4] There exists a shortest optimal Steiner tree T^* for $STP - MSP$ such that every vertex in T^* has degree at most five.*

Lemma 6 *[1] Let T^* be a shortest optimal tree for STP-MSP such that every Steiner point has degree at most five. Let T_j be a full component of T^*. Then the following hold:*

(1) The steinerized minimum spanning tree on terminals in T_j has at most $3 \cdot C(T_j) + 1$ Steiner points.

(2) If T_j contains a Steiner point of degree at most four, then the steinerized minimum spanning tree on terminals in T_j has at most $3 \cdot C(T_j)$ Steiner points.

(3) If the steinerized minimum spanning tree on terminals in T_j has an edge (of length at most R) between two terminals, then it contains at most $3 \cdot C(T_j)$ Steiner points.

From (3) of Lemma 6, we know that if the number of Steiner points contained in a steinerized spanning tree on terminals in a full component T_j reaches the upper bound $3 \cdot C(T_j) + 1$, then any two terminals are not connected directly by a single edge of length at most R, i.e., there must be a Steiner point between them.

Theorem 7 *Let T^* be an optimal tree for STP-MSP and T_A an approximation produced by Algorithm A. Then $C(T_A) \leq 3C(T^*)$.*

Proof. Let T^S be a steinerized minimum spanning tree on all terminals, and let k be the number of 3-stars produced by Step 2 of **Algorithm A**. Then

$$C(T_A) \leq C(T^S) - k.$$

By Lemma 5, we assume that each Steiner point of T^* has degree at most five. Assume that T^* has h full components $T_1, T_2, \ldots, T_h$. For $i = 1, 2$, let $T^{(i)}$ be the components produced by Step i of **Algorithm A**. We construct a steinerized spanning tree T as follows: Initially, set $T := T^{(1)}$, then for each full component T_j $(1 \leq j \leq h)$, add to T the steinerized minimum spanning tree H_j on terminals of T_j, if the resulted tree has a cycle, then destroy the cycle by deleting some edges of H_j. Without loss of generality, suppose that $T_1, T_2, \ldots, T_g$ $(g \leq h)$ are the full components in T^* such that every Steiner point has degree five and $T^{(1)} \cup T_j$ has no cycle. Combining Lemma 4 and Lemma 6 with the fact that for destroying a cycle from $T \cup H_j$, a Steiner point must be removed unless H_j contains an edge between two terminals, we have

$$C(T^S) \leq C(T) \leq 3C(T^*) + g,$$

i.e.,

$$C(T_A) \leq 3C(T^*) + g - k.$$

Suppose that $T^{(1)}$ has p components. Then, $T^{(2)}$ has $p - 2k$ components $C_1, C_2, \ldots, C_{p-2k}$. Now we construct another graph H on all terminals as follows: Initially put all edges of $T^{(1)}$ into H, then consider every T_j $(1 \leq j \leq g)$. If T_j has a unique Steiner point(this Steiner point connects five terminals which must lie in at most two $C_i's$), then among the five terminals there are three pairs (edges) of terminals, each pair (edge) lies in the same C_i. We add the three edges into H. If T_j has at least two Steiner points, then there are two Steiner points each connecting four terminals, and we can also find three pairs (edges) of terminals such that each pair (edge) lies in the same C_i. Thus, we can add the three edges into H. It is clear that H has at most $p-3g$ components. Since each components of H is contained by a C_i, we have $p - 2k \leq p - 3g$, then $g - k \leq \frac{3g}{2} - k \leq 0$. This ends the proof. ∎

4 2.5-Approximation of STP-MSP

In this section, we give a randomized algorithm of ratio-$\frac{5}{2}$ for the STP-MSP problem. Following are some useful Lemmas.

A path $q_1 q_2 \ldots, q_m$ in a tree T is called a *convex* path if for every $i = 1, 2, \ldots, m-3$, $q_i q_{i+2}$ intersects $q_{i+1} q_{i+3}$. An angle of degree more than 120° is called a *big* angle. An angle of degree less than or equal to 120° is called a *small* angle.

Lemma 8 *[1] Let $q_1 q_2 \ldots, q_m$ be a convex path. Suppose there are b big angles among $m-2$ angles $\angle q_1 q_2 q_3, \angle q_2 q_3 q_4, \ldots, \angle q_{m-2} q_{m-1} q_m$. Then, $|q_1 q_m| \leq (b+2)R$.*

Note that if there is no small angle, $|q_1 q_m| \leq (b+1)R$. Thus, this lemma is useful only when there are many small angles in the convex path.

Lemma 9 *[1] In a shortest optimal tree T for $STP - MSP$, there are at most two big angles at a point of degree three, there is at most one big angle at a point of degree four, and there is no big angle with degree five.*

Let T^* be a shortest optimal tree for $STP - MSP$ on n terminals which is a full Steiner tree. Let s_i denote the number of Steiner points of degree i in T^*.

Lemma 10 *[1]* $3s_5 + 2s_4 + s_3 = n - 2$.

Lemma 11 *Consider a clockwise tour F of T^* that visits the n terminals in the order $t_1, t_2, \ldots, t_n, t_1$. (See Figure 3.) Then,*

(i) the tour F has exactly n convex paths $P_1, P_2, \ldots, P_n$ such that P_i connects two terminals t_i and t_{i+1} ($t_{n+1} = t_1$);
(ii) each angle at a Steiner point appears in those n convex paths exactly once.
(iii) connect the two ends of P_i by an edge e_i and then steinerized e_i, $i = 1, 2, \ldots, n$, the total number of Steiner points in any $n-1$ e_i's $\bar{C} = \sum_{i=1}^{n-1} C(e_i)$ is upper bounded as follows:

$$\begin{aligned} \bar{C} &\le s_4 + 2s_3 + 2s_2 + n - C(e_n) \\ &= 3(s_5 + s_4 + s_3) + 2s_2 + 2 - C(e_n) \end{aligned} \tag{5}$$

We denote by T_F the tree consisting of n terminals and $(n-1)$ edges $e_1, e_2, \ldots, e_{n-1}$.

Proof. (i) and (ii) are very easy to see from the structure of T^*. Now, we prove (iii). Consider the tour F. By Lemma 8, if there are a_i big angles in P_i, then there are at most $a_i + 1$ Steiner points on e_i, and so the total number of Steiner points in F is at most n plus the number of big angles in T^*. By Lemma 9, there are at most $2s_2 + 2s_3 + s_4$ big angles in T^*. From Lemma 10, we know that (iii) is valid. ∎

Let T^* be a shortest optimal tree for STP-MSP which is a full Steiner tree on n terminals. Without loss of generality, we assume that T^* has Steiner points

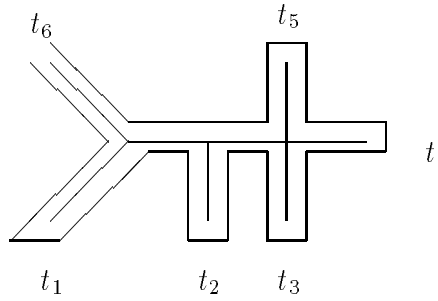


Fig. 3. The tour F that visits all the terminals

of degree at least three. Selecting an arbitrary Steiner point of degree at least three as the root of T^*, we get a rooted tree. A *good point* t in T^* is a Steiner point that is adjacent to some terminals and satisfies one of the following:

(i) (type (1)) t has three or more terminals as children;
(ii) (type (2)) t has two terminals as children and the degree of t is 4;
(iii) (type (3)) t is a point of degree 3.

Note that a good point is of degree at least 3. A *bad* point is a Steiner point of degree at least 3 in T^* that is not a good point.

Theorem 12 *There is a 3-restricted Steiner tree such that each edge has length at most R and the Steiner points is at most $\frac{5}{2}$ times the optimum.*

Now, we focus on the computation of an optimal 3-restricted tree.

Let $H_3(V, F, W)$ be a weighted 3-hypergraph, where $V = P$, $F = \{(a, b)|a \in V \text{ and } b \in V\} \cup \{(a, b, c)|a \in V, b \in V \text{ and } c \in V\}$, and for each edge $e \in F$, $w(e)$ is the smallest number of Steiner points to form an optimal solution of the STP-MSP problem on the terminals in e.

Given three points a, b and c on the on the Euclidean plane, let s be the Steiner point which minimizes $(|sa| + |sb| + |sc|)$, and let k be the number of Steiner points in an optimum solution T of $STP-MSP$ on $\{a, b, c\}$ with constant R.

Lemma 13

$$\lceil \frac{|sa|}{R} \rceil + \lceil \frac{|sb|}{R} \rceil + \lceil \frac{|sc|}{R} \rceil - 2 \geq k \geq \lfloor \frac{|sa|}{R} \rfloor + \lfloor \frac{|sb|}{R} \rfloor + \lfloor \frac{|sc|}{R} \rfloor - 2. \qquad (6)$$

Proof. By steinerizing the optimum Steiner tree, we get a solution of $STP - MSP$ on $\{a, b, c\}$ with exactly $\lceil \frac{|sa|}{R} \rceil + \lceil \frac{|sb|}{R} \rceil + \lceil \frac{|sc|}{R} \rceil - 2$ Steiner points.

Let $|T|$ be the total length of T, which is the sum of the length of edges of T. Then

$$(k - 1) \cdot R + 3R \geq |T| \geq |sa| + |sb| + |sc|,$$

i.e.,

$$k + 2 \geq \frac{|sa| + |sb| + |sc|}{R} \geq \lfloor \frac{|sa| + |sb| + |sc|}{R} \rfloor \geq \lfloor \frac{|sa|}{R} \rfloor + \lfloor \frac{|sb|}{R} \rfloor + \lfloor \frac{|sc|}{R} \rfloor.$$

Therefore, (6) holds. ∎

Lemma 13 gives an upper bound on the cost of (a, b, c).

Lemma 14 *[9] Testing whether three circles has a point in common can be done in constant time.*

For any given points a, b and c on on the Euclidean plane, one can find the minimum Steiner tree on $\{a, b, c\}$ in constant time. Let $q_{a,b,c}$ be the number of Steiner points used to steinerize the optimum Steiner tree on $\{a, b, c\}$, and $q_P = \max\{q_{a,b,c}|\{a, b, c\} \subset P\}$. Then, by Lemma 13 and Lemma 14, the weight W of $H_3(V, F, W)$ can be calculated in $O(n^3 q_P^2)$ time. By Theorem 2 and Theorem 12, we have

$\frac{5}{2}$-approximation Algorithm for STP-MSP

Input A set P of n terminals in the Euclidean plane, a positive constant R
Output A 3-restricted Steiner tree T in which each edge has length at most R.

1. Construct a weighted hypergraph $H_3(V, F, w)$.
2. Call the randomized algorithm in [6] to compute a minimum spanning tree T for $H_3(V, F, w)$;
3. Replace every edge f of the MST T' of $H_3(V, F, w)$ with a Steiner tree with $w(f)$ Steiner points such that the maximum length of each edge in the tree is at most R and output the obtained tree.

Fig. 4. The complete algorithm.

Theorem 15 *Given a set P of n terminals and a positive constant R, there exists a randomized algorithm that computes a solution of $STP - MSP$ on P such that the number of Steiner points is at most $\frac{5}{2}$ times of the optimum running in $poly(n, q_P)$ time with probability at least* 0.5.

The complete algorithm is given in Figure 4.

Acknowledgement

Lusheng Wang is fully supported by HK CERG Grant 9040351. Baogang Xu is supported in part by NSFC 10001035.

References

1. D. Chen, D.-Z. Du, X.. Hu, G. Lin, L. Wang and G. Xue, "Approximations for Steiner trees with minimum number of Steiner points", *Journal of Global Optimization*, vol. 18, pp. 17-33, 2000.
2. C. Chiang, M. Sarrafzadeh and C.K. Wong, "A powerful global router: based on Steiner min-max tree", *IEEE Transactions on Computer-Aided Design*, **19**, pp. 1318-1325, 1990.
3. C.-S. Li, F.F. Tong, C.J. Georgiou and M. Chen, Gain equalization in metropolitan and wide area optical networks using optical amplifiers, *Proc. IEEE INFOCOM'94,* pp. 130-137, June 1994.
4. G. Lin and G. Xue, "Steiner tree problem with minimum number of Steiner points and bounded edge-length", *Information Processing Letters*, **69**, pp. 53-57, 1999.
5. B. Ramamurthy, J. Iness and B. Mukherjee, Minimizing the number of optical amplifiers needed to support a multi-wavelength optical LAN/MAN, *Proc. IEEE INFOCOM'97,* pp. 261-268, April 1997.
6. H. J. Prömel and A. Steger, "A New Approximation Algorithm for the Steiner Tree Problem with Performance Ratio 5/3", *Journal of Algorithms*, **36**, pp. 89-101, 2000.
7. M. Sarrafzadeh and C.K. Wong, "Bottleneck Steiner trees in the plane", *IEEE Transactions on Computers*, **41**, pp. 370-374, 1992.
8. L. Wang and D.-Z. Du, "Approximations for a Bottleneck Steiner Tree Problem", *Algorithmica*, to appear.
9. L. wang and Z. Li, "An Approximation Algorithm for a Bottleneck Steiner Tree Problem in the Euclidean Plane", *Information Processing Letters*, to appear.

An FPTAS for Weight-Constrained Steiner Trees in Series-Parallel Graphs

Guangting Chen[1] and Guoliang Xue[2]

[1] School of Science and Arts,
Hongzhou Institute of Electronic Engineering, Hongzhou, China.
gtchen@mail.hz.zj.cn.
[2] Department of Computer Science,
University of Vermont, Burlington, VT 05405.
xue@cs.uvm.edu.

Abstract. In this paper, we study the problem of computing a minimum cost Steiner tree subject to weight constraint in a series-parallel graph where each edge has a nonnegative integer cost and a nonnegative integer weight. We present a fully polynomial time approximation scheme for this NP-complete problem.

1 Introduction

A computer network is often modeled by an undirected graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. The traditional Steiner tree problem associates an edge cost $c(e) \geq 0$ with each edge $e \in E$ and asks for a minimum cost subgraph of G spanning a given subset $\mathcal{T} \subseteq V$ of *target* vertices. Such problems find important applications in computer networks and have been studied by many researchers [1,4,10,13,15]. We refer the readers to [8,11] for details.

In this traditional network model, there is only one cost corresponding to each edge. In real-life problems, each edge may have a weight besides a cost, and we want to find a minimum cost subgraph of G spanning the vertices in $\mathcal{T}$ with a total weight no more than a given weight constraint $\mathcal{W}$. We call this problem the *weight constrained minimum cost Steiner tree problem* (WCSTP).

Formally, we generalize the traditional network model to allow two independent edge weighting functions: with each edge $e \in E$, there is an associated integer *cost* $c(e) \geq 0$ and an associated integer *weight* $w(e) \geq 0$. Let G' be a subgraph of G. The cost of G', denoted by $c(G')$, is the sum of the edge costs of G'. The weight of G', denoted by $w(G')$, is the sum of the edge weights of G'. Given a set of *target vertices* $\mathcal{T} \subseteq V$ and an integer weight constraint $\mathcal{W} \geq 0$, we are interested in computing a minimum cost tree subgraph T of G which spans the vertices in $\mathcal{T}$ subject to the constraint that $w(T) \leq \mathcal{W}$. We call such a tree a *weight constrained minimum cost Steiner minimum tree*.

When $w(e) = 0$ for every edge $e \in E$, the WCSTP becomes the well known Steiner tree problem. Since the Steiner tree problem is NP-hard in the strong sense [6], the WCSTP is also NP-hard in strong sense. In this paper, we are

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 519–528, 2001.
© Springer-Verlag Berlin Heidelberg 2001

interested in WCSTP on a very important class of sparse networks known as series-parallel graphs (S-P graphs), which are subgraphs of 2-trees defined below.

Following [13], a *2-tree* can be defined recursively as follows, and all 2-trees may be obtained in this way. A triangle (a complete graph on three vertices) is a 2-tree. Given a 2-tree and an edge $\{x, y\}$ of the 2-tree, we can add a new vertex z adjacent to both x and y; the result is a 2-tree. A series-parallel graph (also known as a *partial 2-tree*) is a subgraph of a 2-tree. With this definition, one can see that a 2-tree on n vertices has $2n - 3$ edges and $n - 2$ triangles.

Farley [5] demonstrated that 2-trees are isolated failure immune (IFI) networks. Wald and Colbourn [13] showed that a minimum IFI network is a 2-tree. This fact made 2-trees an important class of fault tolerant networks. Wald and Colbourn [13] also showed that the Steiner tree problem on a series-parallel graph is tractable, presenting a linear time algorithm. In a recent paper, Colbourn and Xue [3] showed that the *grade of service Steiner tree problem* [16] is also tractable on a series-parallel graph.

In this paper, we show that the weight constrained minimum cost Steiner tree problem is NP-hard even when the graph G is a series-parallel graph. We then present a fully polynomial time approximation scheme for computing a weight constrained minimum cost Steiner tree in a series-parallel graph.

2 WCSTP in S-P Graphs Is Intractable

When the number of target vertices is zero or one, the WCSTP is trivial. When there are only two vertices in $\mathcal{T}$, the WCSTP becomes the well-known *weight constrained shortest path problem* (WCSPP) which is known to be NP-hard [6]. To the best of our knowledge, no one has ever addressed the complexity of WCSTP or WCSPP in an S-P graph specifically. In a recent paper [14], Wang and Crowcroft presented an NP-hardness proof for the WCSPP problem in general graphs. We point out that the NP-hardness proof of Wang and Crowcroft, *although targeted at the problem in general graphs*, also proves that the WCSPP is NP-hard in an S-P graph. Since the WCSPP in an S-P graph is NP-hard, the more general WCSTP in an S-P graph is also NP-hard. Therefore we have the following hardness result (for the completeness of the current paper, we include an NP-hardness proof of the WCSPP in an S-P graph in the appendix).

Theorem 1. *The WCSTP in an S-P graph is NP-hard.* □

Note that in the definition of the WCSTP, the weight and cost of an edge are symmetric. Therefore we may also talk about *cost constrained minimum weight Steiner tree problem* (CCSTP) which asks for a minimum weight tree subject to cost constraint. Given the hardness of the problem, we are interested in designing efficient approximation algorithms for this problem. In the next section, we will present a pseudo-polynomial time algorithm for computing a cost constrained minimum weight Steiner tree in an S-P graph. We will then apply standard techniques of scaling and rounding to turn the pseudo-polynomial time algorithm into a fully polynomial time approximation scheme (FPTAS) for weight constrained minimum cost Steiner trees.

3 A Pseudo Polynomial Time Algorithm for CCSTP

In this section, we will present a pseudo polynomial time algorithm for computing a cost constrained minimum weight Steiner tree. Given the set $\mathcal{T} \subseteq V$ and a nonnegative integer ζ, the algorithm computes a minimum weight Steiner tree among those whose cost is bounded by ζ, in $O(n\zeta^2)$ time, where n is the number of vertices in the series-parallel graph.

Our algorithm for computing a weight-constrained Steiner tree operates in two phases. The first completes the graph to a 2-tree, as was done in [13], where each added edge has a weight of zero and a cost that is larger than the cost constraint ζ. This ensures that the added edges will never be chosen in a cost constrained minimum weight Steiner tree.

The second phase involves finding cost constrained minimum weight Steiner trees in 2-trees.

We are given a 2-tree G with target set $\mathcal{T} \subseteq V$. With each (directed) arc $\alpha = (x, y)$ corresponding to an (undirected) edge $\{x, y\}$ of G, we will associate $6\zeta + 6$ measures, which summarize the weight incurred so far in the subgraph S which has been reduced onto the edge $\{x, y\}$:

- $st(\alpha, \xi)$ is the minimum weight of a Steiner tree for S, with a cost of at most ξ, in which x and y appear in the same tree, $\xi = 0, 1, 2, \ldots, \zeta$.
- $dt(\alpha, \xi)$ is the minimum weight of two disjoint trees for S including all targets, one tree involving x and the other involving y, whose total cost is no more than ξ, $\xi = 0, 1, 2, \ldots, \zeta$.
- $yn(\alpha, \xi)$ is the minimum weight of a Steiner tree for S, which includes x but not y, whose cost is at most ξ, $\xi = 0, 1, 2, \ldots, \zeta$.
- $ny(\alpha, \xi)$ is the minimum weight of a Steiner tree for S, which includes y but not x, whose cost is at most ξ, $\xi = 0, 1, 2, \ldots, \zeta$.
- $nn(\alpha, \xi)$ is the minimum weight of a Steiner tree for S, which includes neither x nor y, whose cost is at most ξ, $\xi = 0, 1, 2, \ldots, \zeta$.
- $none(\alpha, \xi)$ is the weight of omitting all vertices of S from a Steiner tree whose cost is at most ξ, $\xi = 0, 1, 2, \ldots, \zeta$ (note that there is a big penalty here is S contains any target vertices).

Let $\beta = (x, y)$ be the other arc corresponding to the edge $\{x, y\}$. Then the $6\zeta + 6$ measures associated with β relates to the measures associated with α in the following way:

$st(\beta, \xi) = st(\alpha, \xi)$	$dt(\beta, \xi) = dt(\alpha, \xi)$	$yn(\beta, \xi) = ny(\alpha, \xi)$
$ny(\beta, \xi) = yn(\alpha, \xi)$	$nn(\beta, \xi) = nn(\alpha, \xi)$	$none(\beta, \xi) = none(\alpha, \xi)$

Let BIG be an integer which is bigger than the sum of all the edge weights in the given graph. Initially, no reduction of the graph has been done. We set the initial measures for each arc $\alpha = (x, y)$ corresponding to an edge $e = \{x, y\}$ as follows, for $\xi = 0, 1, 2, \ldots, \zeta$.

Our algorithm (call it **Algorithm 1**) will use these initial arc measures to recompute the arc measures as the graph is reduced. The graph is reduced by

Table 1. Initialization of arc measures

Initialization Of Measures
$st(\alpha,\xi) = w(e)$ if $c(e) \le \xi$, BIG otherwise;
$dt(\alpha,\xi) = 0$;
$yn(\alpha,\xi) =$ BIG if $y \in \mathcal{T}$, 0 otherwise;
$ny(\alpha,\xi) =$ BIG if $x \in \mathcal{T}$, 0 otherwise;
$nn(\alpha,\xi) =$ BIG if $x \in \mathcal{T}$ or $y \in \mathcal{T}$, 0 otherwise;
$none(\alpha,\xi) =$ BIG if $x \in \mathcal{T}$ or $y \in \mathcal{T}$, 0 otherwise.

repeated deletion of degree-2 vertices. The algorithm stops when there is no degree-2 vertex left. Suppose at some point in the algorithm there is a triangle $\{x, y, z\}$ in which y is a degree-2 vertex and that costs have been computed for both (x, y) and (y, z). Let $L = (x, y)$, $R = (y, z)$, $M = (x, z)$. We will update the measures at M using the following rules:

– For $\xi = 0, 1, 2, \ldots, \zeta$, let $w_{\xi,st}$ be the minimum of the following four sets of numbers:

$\{st(M,\xi_M) + st(L,\xi_L) + dt(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{st(M,\xi_M) + dt(L,\xi_L) + st(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{st(M,\xi_M) + yn(L,\xi_L) + ny(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{dt(M,\xi_M) + st(L,\xi_L) + st(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$.

– For $\xi = 0, 1, 2, \ldots, \zeta$, let $w_{\xi,dt}$ be the minimum of the following three sets of numbers:

$\{dt(M,\xi_M) + st(L,\xi_L) + dt(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{dt(M,\xi_M) + dt(L,\xi_L) + st(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{dt(M,\xi_M) + yn(L,\xi_L) + ny(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$.

– For $\xi = 0, 1, 2, \ldots, \zeta$, let $w_{\xi,yn}$ be the minimum of the following two sets of numbers:

$\{yn(M,\xi_M) + yn(L,\xi_L) + none(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{yn(M,\xi_M) + st(L,\xi_L) + yn(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$.

– For $\xi = 0, 1, 2, \ldots, \zeta$, let $w_{\xi,ny}$ be the minimum of the following two sets of numbers:

$\{ny(M,\xi_M) + ny(L,\xi_L) + st(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{ny(M,\xi_M) + none(L,\xi_L) + ny(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$.

– For $\xi = 0, 1, 2, \ldots, \zeta$, let $w_{\xi,nn}$ be the minimum of the following four sets of numbers:

$\{nn(M,\xi_M) + none(L,\xi_L) + none(R,\xi_R) | \xi_L + \xi_R + \xi_M = \xi, \xi_L \ge 0, \xi_R \ge 0, \xi_M \ge 0\}$;

$\{none(M,\xi_M)+nn(L,\xi_L)+none(R,\xi_R)|\xi_L+\xi_R+\xi_M=\xi,\xi_L\geq 0,\xi_R\geq 0,\xi_M\geq 0\}$;

$\{none(M,\xi_M)+none(L,\xi_L)+nn(R,\xi_R)|\xi_L+\xi_R+\xi_M=\xi,\xi_L\geq 0,\xi_R\geq 0,\xi_M\geq 0\}$;

$\{none(M,\xi_M)+ny(L,\xi_L)+yn(R,\xi_R)|\xi_L+\xi_R+\xi_M=\xi,\xi_L\geq 0,\xi_R\geq 0,\xi_M\geq 0\}$.

– For $\xi=0,1,2,\ldots,\zeta$, let $w_{\xi,none}$ be the minimum of the following set of numbers:

$\{none(M,\xi_M)+none(L,\xi_L)+none(R,\xi_R)|\xi_L+\xi_R+\xi_M=\xi,\xi_L\geq 0,\xi_R\geq 0,\xi_M\geq 0\}$.

– Delete the vertex y from the graph. For $\xi=0,1,2,\ldots,\zeta$, update the values of $st(M,\xi)$, $dt(M,\xi)$, $yn(M,\xi)$, $ny(M,\xi)$, $nn(M,\xi)$, and $none(M,\xi)$ to $w_{\xi,st}$, $w_{\xi,dt}$, $w_{\xi,yn}$, $w_{\xi,ny}$, $w_{\xi,nn}$, and $w_{\xi,none}$, respectively.

Theorem 2. *When the 2-tree is reduced to a single edge $\alpha=\{x,y\}$, the minimum of the four values $st(\alpha,\zeta)$, $yn(\alpha,\zeta)$, $ny(\alpha,\zeta)$, $nn(\alpha,\zeta)$ is the minimum weight of a Steiner tree interconnecting all the vertices in $\mathcal{T}$ with a cost not more than ζ, where a weight of* BIG *or more indicates the nonexistence of a cost constrained Steiner tree.*

Proof. We note that the measures of an edge $\alpha=\{x,y\}$ are initialized correctly, and are updated correctly after each reduction. For example, for any $\xi\in\{0,1,2,\ldots,\zeta\}$ and an edge $\alpha=\{x,y\}$, $st(\alpha,\xi)$ is the minimum weight of a Steiner tree for S whose cost is no more than ξ, in which x and y appear in the same tree.

When the graph is reduced to a single arc $\alpha=\{x,y\}$, $st(\alpha,\zeta)$ is the minimum weight of a cost constrained Steiner tree interconnecting x, y, and all the vertices in $\mathcal{T}$; $yn(\alpha,\zeta)$ is the minimum weight of a cost constrained Steiner tree interconnecting x (but not y) and all the vertices in $\mathcal{T}-\{y\}$; $ny(\alpha,\zeta)$ is the minimum weight of a cost constrained Steiner tree interconnecting y (but not x) and all the vertices in $\mathcal{T}-\{x\}$; $nn(\alpha,\zeta)$ is the minimum weight of a cost constrained Steiner tree interconnecting y (but not x) and all the vertices in $\mathcal{T}-\{x,y\}$. Note that leaving out a target vertex from the tree receives a penalty of at least BIG. Therefore the minimum of the above four values corresponds to the minimum weight of a cost constrained Steiner tree. □

Theorem 3. *Algorithm 1 can be implemented in $O(n\zeta^2)$ time, where ζ is the cost constraint and n is the number of vertices in the graph.*

Proof. We will show that the update of measurements can be accomplished in $O(\zeta^2)$ time for each reduction. Since there are $O(n)$ reductions, this would complete the proof of the theorem.

Since there are $O(\zeta)$ choices for each of ξ, ξ_M and ξ_L ($\xi_R=\xi-\xi_M-\xi_L$), a simple minded implementation would require $O(\zeta^3)$ time to update the measures for each reduction.

Let us denote $\xi_L + \xi_R$ by ξ_{LR}. Although there are $O(\zeta^2)$ different choices for (ξ_L, ξ_R), there are only $O(\zeta)$ different choices for ξ_{LR}. Using $O(\zeta^2)$ time, we can compute the following $12(\zeta+1)$ measures for all $\xi_{LR} \in \{0, 1, 2, \ldots, \zeta\}$.

$$\begin{aligned}
LR(\xi_{LR}, st, st) &= \min\{st(L, \xi_L) + st(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, st, dt) &= \min\{st(L, \xi_L) + dt(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, dt, st) &= \min\{dt(L, \xi_L) + st(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, yn, ny) &= \min\{yn(L, \xi_L) + ny(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, yn, none) &= \min\{yn(L, \xi_L) + none(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, st, yn) &= \min\{st(L, \xi_L) + yn(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, ny, st) &= \min\{ny(L, \xi_L) + st(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, none, ny) &= \min\{none(L, \xi_L) + ny(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, none, none) &= \min\{none(L, \xi_L) + none(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, nn, none) &= \min\{nn(L, \xi_L) + none(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, none, nn) &= \min\{none(L, \xi_L) + nn(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\};\\
LR(\xi_{LR}, ny, yn) &= \min\{ny(L, \xi_L) + yn(R, \xi_{LR} - \xi_L) | 0 \le \xi_L \le \xi_{LR}\}.
\end{aligned}$$

After those $12(\zeta+1)$ measures are computed, only $O(\zeta^2)$ time are necessary to compute the updated values for $st(M, \xi)$, $dt(M, \xi)$, $yn(M, \xi)$, $ny(M, \xi)$, $nn(M, \xi)$, $none(M, \xi)$, for $\xi = 0, 1, 2, \ldots, \zeta$. For example, $w_{\xi, yn}$ can be computed as the minimum of

$$\min\{yn(M, \xi_M) + LR(\xi - \xi_M, yn, none) | 0 \le \xi_M \le \xi\}$$

and

$$\min\{yn(M, \xi_M) + LR(\xi - \xi_M, st, yn) | 0 \le \xi_M \le \xi\}.$$

This shows that Algorithm 1 can be implemented in $O(n\zeta^2)$ time. □

We point out that the weight constrained Steiner tree can be constructed in $O(n\zeta)$ extra time time if we perform some bookkeeping operations (recording how the minimum was achieved) during the reductions.

4 An FPTAS for WCSTP

We use standard techniques of scaling and rounding [2,7,9,12,17] to turn the pseudo polynomial time algorithm for CCSTP into a fully polynomial time approximation scheme for WCSTP.

Let us use $c(\mathcal{T}, \mathcal{W})$ to denote the minimum cost of a Steiner tree spanning the targets in $\mathcal{T}$ with a weight of no more than $\mathcal{W}$. Given a positive real number $\mathcal{C}$, deciding whether $c(\mathcal{T}, \mathcal{W}) > \mathcal{C}$ or $c(\mathcal{T}, \mathcal{W}) \le \mathcal{C}$ is NP-hard. Using the standard technique of scaling and rounding [2,7,9,12,17], we can decide, in fully polynomial time, whether $c(\mathcal{T}, \mathcal{W}) > \mathcal{C}$ or $c(\mathcal{T}, \mathcal{W}) < (1+\epsilon)\mathcal{C}$, for any given constant $\epsilon > 0$. This technique will play an important role in our FPTAS for computing

a weight constrained minimum cost Steiner tree in an S-P graph. We describe this approximate testing in Algorithm 2 as TEST .

Algorithm 2 $\mathsf{TEST}(\mathcal{C}, \epsilon)$

Step_1 Set $\theta := \frac{2n-3}{\mathcal{C} \times \epsilon}$; Let c_θ be the *scaled edge cost function* such that $c_\theta(e) = \lfloor c(e) \times \theta \rfloor$ for $e \in E$; Set $\zeta := \mathcal{C} \times \theta$;

Step_2 Apply Algorithm 1 using the scaled edge cost function c_θ instead of the original edge cost function c;
if the weight of the cost constrained Steiner tree is no more than $\mathcal{W}$ then
output YES ;
else
output NO ;
endif

Theorem 4. *Let us be given the target set $\mathcal{T}$, the weight constraint $\mathcal{W}$, the positive real numbers $\mathcal{C}$ and ϵ. If $\mathsf{TEST}(\mathcal{C}, \epsilon) = \mathsf{NO}$, then $c(\mathcal{T}, \mathcal{W}) > \mathcal{C}$. If $\mathsf{TEST}(\mathcal{C}, \epsilon) = \mathsf{YES}$, then $c(\mathcal{T}, \mathcal{W}) < (1+\epsilon) \times \mathcal{C}$. In addition, the worst-case time complexity of $\mathsf{TEST}(\mathcal{C}, \epsilon)$ is $O(\frac{n^3}{\epsilon^2})$.*

PROOF. Let T be a tree subgraph in G. Let $c(T) = \sum_{e \in T} c(e)$ and $c_\theta(T) = \sum_{e \in T} c_\theta(e)$. Since the number of edges in T is at most $2n-3$, we can prove that $c_\theta(T) \leq \zeta$ implies $c(T) \leq \mathcal{C}(1+\epsilon)$ and that $c_\theta(T) > \zeta$ implies $c(T) > \mathcal{C}$, where θ and ζ are as defined in Algorithm 2.

Assume that $\mathsf{TEST}(\mathcal{C}, \epsilon) = \mathsf{NO}$. Then we know that for any tree T spanning the targets in $\mathcal{T}$ with a weight no more than $\mathcal{W}$, we must have $c_\theta(T) > \zeta$, which in turn implies $c(T) > \mathcal{C}$. This says that $c(\mathcal{T}, \mathcal{W}) > \mathcal{C}$.

Now assume that $\mathsf{TEST}(\mathcal{C}, \epsilon) = \mathsf{YES}$. Then we know that there is a tree T spanning the targets in $\mathcal{T}$ such that $w(T) \leq \mathcal{W}$ and $c_\theta(T) \leq \zeta$. Note that $c_\theta(T) \leq \zeta$ implies $c(T) < \mathcal{C}(1+\epsilon)$. This says that $c(\mathcal{T}, \mathcal{W}) > \mathcal{C}(1+\epsilon)$.

The time complexity of TEST follows directly from Theorem 3. □

We will use LB and UB to denote lower and upper bounds on $c(\mathcal{T}, \mathcal{W})$. Our FPTAS starts with efficiently computable values of LB and UB and uses bisection to reduce the ratio $\frac{\mathsf{UB}}{\mathsf{LB}}$.

An initial value of LB can be computed as follows. Use Algorithm 1 to compute a minimum weight Steiner tree T whose cost is 0. This clearly can be done in polynomial time. If the weight of this tree is no more than $\mathcal{W}$, it is also a weight constrained minimum cost Steiner tree and we are done. Otherwise, $c(\mathcal{T}, \mathcal{W}) \geq 1$ and we can use 1 as the initial value of LB.

An initial value of UB can also be computed easily. Use the algorithm in [13] to compute a minimum weight (ignore the cost) Steiner tree T spanning the target vertices in $\mathcal{T}$. If $w(T) > \mathcal{W}$, there is no Steiner tree whose weight is bounded by $\mathcal{W}$. When $w(T) \leq \mathcal{W}$, we may use $c(T)$ as the initial value of UB.

Let $\mathbf{B}$ be some chosen real number which is grater than $1+\epsilon/3$. We will apply bisection to drive the ratio $\frac{\mathsf{UB}}{\mathsf{LB}}$ down to some number below $\mathbf{B}$. Suppose that our lower bound LB and upper bound UB are such that $\mathsf{UB} > \mathsf{LB} \times \mathbf{B} > \mathsf{LB} \times (1+\epsilon/3)$. Let $\mathcal{C} = \sqrt{\frac{\mathsf{UB} \times \mathsf{LB}}{1+\epsilon/3}}$. If $\mathsf{TEST}(\mathcal{C}, \epsilon/3) = \mathsf{NO}$, then $\mathcal{C}$ is also a lower bound for $c(\mathcal{T})$.

If $\mathsf{TEST}(\mathcal{C}, \epsilon/3) = \mathsf{YES}$, then $(1 + \epsilon/3) \times \mathcal{C}$ is also an upper bound for $c(\mathcal{T})$. Therefore, the ratio of the new upper bound over the new lower bound will be no more than $\sqrt{\frac{\mathsf{UB}}{\mathsf{LB}} \times (1 + \epsilon/3)}$. Let us call the above process an *iteration*. Note that such an iteration can be accomplished in fully polynomial time (by Theorem 4). Furthermore, the ratio of the upper-bound over the lower bound can be reduced to a number below $\mathbf{B}$ in polynomial number of iterations (polynomial in the input size of the given instance and $\frac{1}{\epsilon}$). This analysis leads to out FPTAS which is given in Algorithm 3.

Algorithm 3 FPTAS for weight constrained minimum cost Steiner tree in an S-P graph.

Step_1 set $\mathbf{B} := (1 + \epsilon/3) \times (1 + \epsilon/3)$;
set LB and UB to their initial values;
Step_2 if $\mathsf{UB} \leq \mathbf{B} \times \mathsf{LB}$ then
goto Step_3;
else
let $\mathcal{C} := \sqrt{\frac{\mathsf{UB} \times \mathsf{LB}}{1+\epsilon/3}}$;
if $\mathsf{TEST}(\mathcal{C}, \epsilon/3) = \mathsf{NO}$, set $\mathsf{LB} = \mathcal{C}$;
if $\mathsf{TEST}(\mathcal{C}, \epsilon/3) = \mathsf{YES}$, set $\mathsf{UB} = \mathcal{C} \times (1 + \epsilon/3)$;
goto Step_2;
endif
Step_3 set $\theta := \frac{2n-3}{\mathsf{LB} \times \epsilon/3}$; $\zeta := \theta \times \mathsf{UB}$;
set $c_\theta(e) := \lfloor \theta \times c(e) \rfloor$ for every $e \in E$;
apply Algorithm 1 to compute a cost constrained minimum weight Steiner tree using the scaled cost function c_θ.

Theorem 5. *If there is no weight constrained Steiner tree spanning the target vertices in $\mathcal{T}$, we will find this out during our computation of the initial value of* UB*. If $\mathcal{T}$ has a weight constrained Steiner tree, Algorithm 3 finds a weight constrained Steiner tree T such that $w(T) \leq \mathcal{W}$ and $c(T) \leq (1 + \epsilon) \times c(\mathcal{T}, \mathcal{W})$. Furthermore, the time complexity of Algorithm 3 is $O(\frac{n^3}{\epsilon^2} \times (\log \mathcal{S} + \log(\frac{1}{\epsilon}))$, where $\mathcal{S}$ is the input size of the given instance.*

Proof. The claim follows from standard techniques [2,7,17] and Theorems 2-4. □

5 Conclusions

In this paper, we have studied the weight constrained minimum cost Steiner tree problem on a very important class of sparse graphs–series-parallel graphs. Although most Steiner tree problems are tractable on series-parallel graphs, it is shown that the weight constrained minimum cost Steiner tree problem on series-parallel graphs is NP-hard. On the positive side, we have presented a fully polynomial time approximation scheme for this problem, which has applications to computer networks as well as computational biology. A more challenging problem is the weight constrained minimum cost Steiner tree problem on a general graph.

Acknowledgments

Research of the authors is supported by ARO grant DAAD19-00-1-0377 (Xue), DOE grant DE-FG02-00ER45828 (Xue), Chinese NSF grant CNSF19971078 (Chen) and Chinese Institute of Electronic Science grant 57-7-3-7 (Chen).

References

1. S. Arora, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, *Journal of the ACM*, Vol. 45(1998), pp. 753–782.
2. G.T. Chen and G.L. Xue, K-pair delay constrained minimum cost routing in undirected networks, *SODA2001–ACM-SIAM Symposium on Discrete Algorithms*, pp. 230–231.
3. C.J. Colbourn and G. Xue, Grade of service Steiner trees on a series-parallel network, in D.-Z. Du, J.M. Smith, and J.H. Rubinstein (Eds.), *Advances in Steiner Trees*, Kluwer Academic Publishers, 2000, pp. 163–174.
4. D.Z. Du and F.K. Hwang, An approach for proving lower bounds: solution of Gilbert-Pollak conjecture on Steiner ratio, *IEEE FOCS'90*, pp. 76–85.
5. A.M. Farley, Networks immune to isolated failures, *Networks*, Vol. 11(1981), pp. 255–268.
6. M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H Freeman and Company, 1979.
7. R. Hassin, Approximation schemes for the restricted shortest path problem, *Mathematics of Operations Research*, Vol. 17(1992), pp. 36–42.
8. F.K. Hwang, D.S. Richards and Pawel Winter, *The Steiner tree problem*, *Annals of Discrete Mathematics*, Vol. 53, North-Holland, 1992.
9. O. Ibarra and C. Kim, Fast approximation algorithms for the knapsack and sum of subsets problems, *Journal of ACM*, Vol. 22, pp. 463–468.
10. X. Jia and L. Wang, Group multicast routing using multiple minimum Steiner trees, *Journal of Computer Communications*, pp. 750–758, 1997.
11. T. Jiang and L. Wang, Computing shortest networks under a fixed topology, in D.-Z. Du, J.M. Smith, and J.H. Rubinstein (Eds.), *Advances in Steiner Trees*, Kluwer Academic Publishers, 2000, pp. 39–62.
12. S. Sahni, General techniques for combinatorial approximations, *Operations Research*, Vol. 35(1977), pp. 70-79.
13. J.A. Wald and C.J. Colbourn, Steiner trees, partial 2-trees, and minimum IFI networks, *Networks*, Vol. 13(1983), pp. 159–167.
14. Zheng Wang and Jon Crowcroft, Quality-of-service routing for supporting multimedia applications, *IEEE Journal on Selected Areas in Communications*, Vol. 14(1996), pp. 1228–1234.
15. P. Winter, Generalized Steiner problem in series-parallel networks, *J. Algorithms*, Vol. 7(1986), pp. 549-566.
16. G.L. Xue, G.H. Lin and D.Z. Du, Grade of service Steiner minimum trees in the Euclidean plane, *Algorithmica*, accepted for publication.
17. G.L. Xue and W. Xiao, An FPTAS for minimum cost delay constrained multicast tree under a Steiner topology, with applications, submitted for publication.

A NP-Completeness of WCSTP in S-P Graphs

In this section, we prove that the weight constrained shortest path problem [7] in S-P graphs is NP-hard. After we have arrived at this proof, we found that the proof by Wang and Crowcroft [14] also proves the same fact. This proof is included here as an aid to the referees.

Theorem 6. *The weight constrained shortest path problem in series-parallel graphs is NP-hard.*

Proof. Now we transform PARTITION to the weight constrained shortest path problem in S-P graphs.

Suppose we have a set S with n elements and every element a has a corresponding size $s(a) \in Z^+$, $\sum_{a \in S} s(a) = 2A$. We construct a S-P graph (see Figure 1) as follows.

- $V = \{v_1, v_2, \ldots, v_{n+1}, u_1, u_2, \ldots, u_n\}$,
- $E = \{(v_i, v_{i+1}) | i = 1, 2, \ldots, n\} \cup \{(v_i, u_i), (v_{i+1}, u_i) | i = 1, 2, \ldots, n\}$,
- For $e = (v_i, v_{i+1}), c(e) = s(a_i), w(e) = 0$.
- For $e = (v_i, u_i)$ or (v_{i+1}, u_i), $c(e) = 0, w(e) = \frac{1}{2}s(a_i)$.

For this S-P graph, we want to find a shortest path from v_1 to v_{n+1} such that the total weight of this path is no larger than A.

If we can find a partition of $S = S_1 \cup S_2$ such that $\sum_{a \in S_1} s(a) = \sum_{a \in S_2} s(a) = A$, then we have a path from v_1 to v_{n+1} consist of the edges (v_i, v_{i+1}) for $a_i \in S_1$, and $(v_i, u_i), (u_i, v_{i+1})$ for $a_i \in S_2$. The total cost and total weight of this path are all equal to A. This path is obviously the shortest path under the restriction of total weight no larger than A.

Conversely, assume that we have find a shortest path from v_1 to v_{n+1} with the total weight no larger than A. From the construction of the S-P graph, we can get a partition of S by setting $S_1 = \{a_i | (v_i, v_{i+1})$ is in the shortest path$\}$.

From above discussion, we know that the weight constrained shortest path problem in S-P graph is NP-hard. □

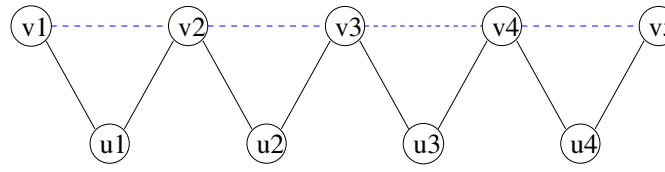


Fig. 1. Hardness proof of WCSPP.

Decidable Approximations on Generalized and Parameterized Discrete Timed Automata

Zhe Dang[1], Oscar H. Ibarra[2], and Richard A. Kemmerer[2]

[1] School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164
zdang@eecs.wsu.edu
[2] Department of Computer Science
University of California, Santa Barbara, CA 93106
{ibarra,kemm}@cs.ucsb.edu

Abstract. We consider generalized discrete timed automata with general linear relations over clocks and parameterized constants as clock constraints and with parameterized durations. We look at three approximation techniques (i.e., the r-reset-bounded approximation, the B-bounded approximation, and the $\langle B, r\rangle$-crossing-bounded approximation), and derive automata-theoretic characterizations of the binary reachability under these approximations. The characterizations allow us to show that the safety analysis problem is decidable for generalized discrete timed automata with unit durations and for deterministic generalized discrete timed automata with parameterized durations. An example specification written in ASTRAL is used to run a number of experiments using one of the approximation techniques.

1 Introduction

As a standard model for analyzing real-time systems, timed automata [3] have received enormous attention during the past decade. A fundamental result in the theory of timed automata shows that region reachability for timed automata is decidable [3]. This result has been very useful in defining various real-time logics, appropriate model checking algorithms and tools [2,4,16,17,24,25,27,28,29] for verifying real-time systems (see [1] for a survey).

However, not every real-time system can be modeled as a timed automaton since a complex (not necessarily large) real-time system might contain non-region (e.g., Presburger) clock constraints. However, the "Turing computing" power of such augmented automata prevents automatic verification of properties such as reachability [3,5]. Therefore, decidable approximation techniques are of great interest, since these techniques would provide a user some degree of confidence in analyzing and debugging complex real-time specifications. In contrast to the most direct approximation techniques [5,10,11,12] that bound the number of transitions to a fixed number, the approximation techniques presented in this paper restrict the clock behaviors but do not necessarily bound the number of transition iterations to be finite.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 529–539, 2001.
© Springer-Verlag Berlin Heidelberg 2001

In this paper, we focus on timed automata with integer-valued clocks, i.e., discrete timed automata, and extend them to generalized discrete timed automata by allowing general linear relations over clocks and parameterized constants as clock constraints. Furthermore, the duration of a transition can be a parameterized constant. These generalizations have practical motivation. For example, many complex real-world specifications [6,8,10,22] written in the real-time specification language ASTRAL [6] use generalized clock constraints and parameterized durations in almost every specification. Therefore, the results presented in this paper may be useful in implementing an automatic specification debugger for complex real-time specification languages like ASTRAL.

We investigate three approximation techniques in this paper. The r-reset-bounded approximation bounds the number of clock resets by a given positive integer r for each clock. The B-bounded approximation requires that each clock reset happens before its value exceeds a given (integral) time B. Combining these two, the $\langle B, r\rangle$-crossing-bounded approximation requires that for each clock, there are at most r times that the clock resets after its value exceeds B. Given an approximation technique, we will focus on the binary reachability characterization of a generalized discrete timed automaton. Binary reachability has recently been proposed and investigated for a number of timed systems [7,9,14,15], which makes it possible to reason about "non-region" properties for these systems. We first show that, when a generalized discrete timed automaton has unit duration, the binary reachability under any one of the three approximations is Presburger. Then by considering a generalized discrete timed automaton with parameterized durations, we show that the binary reachability under any one of the three approximations has a 2DCM-padding when the machine is deterministic. Specifically, we show that the "padded language" for binary reachability can be accepted by a deterministic two-way counter machine with one reversal-bounded counter [20]. The case for nondeterministic generalized discrete timed automata is open. These are good characterizations in the sense that the validity of Presburger formulas as well as the emptiness problem for these counter machines are decidable. This allows us to establish, in principle, decidable results for the (Presburger) safety analysis problem for generalized discrete timed automata under the approximations. The 2DCM-padding characterization is particularly interesting, since binary reachability is not necessarily semilinear.

Due to space limitation, all the proofs are omitted. For a complete exposition see [13].

2 Preliminaries

Let V be a finite set of variables over the integers. An *atomic linear relation* on V is defined as $\Sigma_{v\in V} a_v v < b$, where a_v and b are integers. A *linear relation* on V is constructed from a finite number of atomic linear relations using $\neg$ and $\wedge$. $\mathcal{L}_V$ denotes the set of all linear relations on V. An *atomic Presburger relation* on V is either an atomic linear relation on V or a linear congruence $\Sigma_{v\in V} a_v v = b \bmod d$, where a_v, b and d are integers. A Presburger formula can always be constructed

from atomic Presburger relations using $\neg$ and $\wedge$. Let N be the set of integers with N^+ denoting the set of nonnegative integers. A *clock* is a variable over N^+. A *generalized discrete timed automaton* $\mathcal{A}$ is a tuple $\langle S, \boldsymbol{C}, \boldsymbol{X}, E\rangle$ where S is a finite set of *(control) states*, $\boldsymbol{X}$ is a finite set of clocks, $\boldsymbol{C}$ is a finite set of parameterized constants, and $E \subseteq S \times (\boldsymbol{C} \cup \{0,1\}) \times 2^{\boldsymbol{X}} \times \mathcal{L}_{\boldsymbol{X}\cup\boldsymbol{C}} \times S$ is a finite set of edges. Each edge $\langle s, d, \lambda, l, s'\rangle$ denotes a *transition (or an edge)* from s to s' with *enabling condition* $l \in \mathcal{L}_{\boldsymbol{X}\cup\boldsymbol{C}}$. $d \in \boldsymbol{C} \cup \{0,1\}$ is a parameterized constant indicating the duration of this transition. $\lambda \subseteq \boldsymbol{X}$ is the set clocks that are reset as a result of this transition. When λ is empty, the duration, which is a parameterized constant in $\boldsymbol{C}$ or integer constant 1, must be positive. Clock resets take no time. Thus, when λ is not empty, the duration d on this edge is simply 0.

The semantics is defined as follows. $\alpha \in S \times (N^+)^{|\boldsymbol{C}|} \times (N^+)^{|\boldsymbol{X}|}$ is called a *configuration* with α_q being the state under this configuration. α_{x_i} and α_{c_j} denote the value of the clock x_i and the value of the parameterized constant c_j, respectively. Note that each clock and parameterized constant are nonnegative, i.e., in N^+. $\alpha \rightarrow^{\langle s,d,\lambda,l,s'\rangle} \alpha'$ denotes a one-step transition along an edge in $\mathcal{A}$ satisfying

- Constant values do not change, i.e., for each $c \in \boldsymbol{C}$, $\alpha_c = \alpha'_c$,
- The state s is set to a new location s', i.e., $\alpha_q = s, \alpha'_q = s'$,
- Each clock changes according to the edge given. When there is no clock reset on this edge, i.e., $\lambda = \emptyset$, all the clocks synchronously progress by the amount of the duration, i.e., for each $x \in \boldsymbol{X}$, $\alpha'_x = \alpha_x + \alpha_d$. In this case, the duration is positive, i.e., $\alpha_d > 0$. When $\lambda \neq \emptyset$, clocks in λ reset to 0 and all the other clocks are unchanged. Thus, clock resets take no time. That is, for each $x \in \lambda$, $\alpha'_x = 0$ and for each $x \notin \lambda$, $\alpha'_x = \alpha_x$.
- The enabling condition is satisfied. That is, $l(\alpha)$ is true.

We simply write $\alpha \rightarrow \alpha'$ if α can reach α' by a one-step transition. $\mathcal{A}$ is *deterministic* if for any configuration α there is at most one β such that $\alpha \rightarrow \beta$. A *path* $\alpha_0 \cdots \alpha_k$ satisfies $\alpha_i \rightarrow \alpha_{i+1}$ for each i. Write $\alpha \rightsquigarrow_{\mathcal{A}} \beta$ if α reaches β through a path. In the definition of $\mathcal{A}$, there is no input. This is because the input is always one-way for timed automata and, therefore, input symbols can be built into the control states. $\mathcal{A}$ has parameterized durations. If we restrict each duration on an edge without clock resets to be 1, then $\mathcal{A}$ is called a generalized discrete timed automaton *with unit durations*.

When $\mathcal{A}$ with unit durations contains no parameterized constants and enabling conditions are *clock constraints* or *clock regions* in the form of Boolean combinations of $x - y\#c$ and $x\#c$, where c is an integer, x and y are clocks and $\# \in \{<, =, >\}$, $\mathcal{A}$ is equivalent to the standard timed automata (with integer-valued clocks) [3]. On the other hand, when $\mathcal{A}$ with unit durations contains enabling conditions only in the form of Boolean combinations of $x\#c$, where c is a parameterized constant or an integer, x is a clock (when c is a parameterized constant, x is called a *parameterized clock*), $\mathcal{A}$ is a *parameterized timed automaton* (with integer-valued clocks) [5].

There are two kinds of reachability for $\mathcal{A}$. They are state reachability and binary reachability. Assume a state s_0 is designated as the *initial state* of $\mathcal{A}$. A state $s \in S$ is *state reachable* in $\mathcal{A}$ if there is an initial configuration α_0 (whose state is s_0 and whose clock values are all 0) and a configuration α with $\alpha_q = s$ such that α is reachable from the initial configuration, i.e., $\alpha_0 \rightsquigarrow_{\mathcal{A}} \alpha$. The state reachability set is the set of all states s such that s is state reachable in $\mathcal{A}$. The *state reachability problem* of $\mathcal{A}$ is whether a state $s \in S$ is state reachable. The following are some known results about the state reachability problem.

Theorem 1. *(1). The state reachability problem is decidable for standard timed automata [3]. (2). The state reachability problem is decidable for parameterized timed automata with only one parameterized clock (but can have many unparameterized clocks) [5]. (3). The state reachability problem is undecidable for generalized discrete timed automata.*

Actually, Theorem 1 (3) follows from the following special cases.

Theorem 2. *(1). The state reachability problem is undecidable for standard timed automata when we allow "+" operations in clock constraints, e.g., $x + y - z < 5$ [3]. (2). The state reachability problem is undecidable for parameterized timed automata (with more than 2 parameterized clocks) [5].*

On the other hand, *binary reachability* is the set of all configuration pairs $\langle \alpha, \beta \rangle$ such that $\alpha \rightsquigarrow_{\mathcal{A}} \beta$ (we use $\rightsquigarrow_{\mathcal{A}}$ in the paper). Characterizations of the binary reachability of timed automata have recently been established.

Theorem 3. *(1). The binary reachability of timed automata with real valued clocks is definable in the additive theory over reals and integers [7,9]. (2). The binary reachability of timed automata with integer valued clocks is definable by a Presburger formula [7,15].*

Characterizations of binary reachability help us to reason about non-region properties of timed systems. For instance, consider the following property: "for every configuration α there exists a configuration β such that $\alpha \rightsquigarrow_{\mathcal{A}} \beta$ and the clock x_1 in β is the sum of clocks x_1 and x_2 in α." Though constraint $\beta_{x_1} = \alpha_{x_1} + \alpha_{x_2}$ is not in the form of a clock region, this property can be automatically verified for timed automata [7,9,15].

However, for generalized discrete timed automata, the binary reachability $\rightsquigarrow_{\mathcal{A}}$ is too strong to have an interesting characterization. In particular, even the membership problem for binary reachability, i.e., deciding whether two given configurations α and β satisfy $\alpha \rightsquigarrow_{\mathcal{A}} \beta$ for a generalized discrete timed automaton $\mathcal{A}$, is undecidable. This follows from the fact that a two-counter machine can be simulated by a generalized discrete timed automaton $\mathcal{A}$, as shown in the the proofs of Theorem 2 (1) (2) [3,5]. Thus, the membership problem can be reduced to the halting problem for two-counter machines, which is undecidable.

Theorem 4. *The membership problem for binary reachability is undecidable for generalized discrete timed automata.*

This undecidability result for generalized discrete timed automata leads us to consider the following three approximations of $\rightsquigarrow_{\mathcal{A}}$. Let r and B be fixed positive integers. The first approximation is *r-reset-bounded* reachability. A path $\alpha^0\alpha^1\cdots\alpha^k$ is called *r-reset-bounded* if each clock resets at most r times. Write $\alpha \rightsquigarrow^r_{\mathcal{A}} \beta$ if α reaches β through an r-reset-bounded path. The second approximation is *B-bounded* reachability. A path $\alpha^0\alpha^1\cdots\alpha^k$ is called *B-bounded* if for each $j < k$, each $x_i \in \boldsymbol{X}$, $|\alpha^j_{x_i} - \alpha^{j+1}_{x_i}| < B$. Write $\alpha \rightsquigarrow^B_{\mathcal{A}} \beta$ if α reaches β through a B-bounded path. The third approximation is *$\langle B, r\rangle$-crossing-bounded* reachability. A path $\alpha^0\alpha^1\cdots\alpha^k$ is called *$\langle B, r\rangle$-crossing-bounded* if there are at most r many i's such that $|\alpha^j_{x_i} - \alpha^{j+1}_{x_i}| \geq B$. Write $\alpha \rightsquigarrow^{\langle B,r\rangle}_{\mathcal{A}} \beta$ if α reaches β through a $\langle B, r\rangle$-crossing-bounded path.

The main results in this paper show that the three approximations of binary reachability $\rightsquigarrow_{\mathcal{A}}$ have decidable characterizations, i.e., they can be accepted by a class of machines with a decidable emptiness problem. Before we proceed to show the results, some further definitions are needed.

A nondeterministic multicounter machine (NCM) is a nondeterministic machine with a finite number of states, a one-way input tape, and a finite number of integer counters. Each counter can be incremented by 1, decremented by 1, or stay unchanged. These counter assignments are called *standard assignments*. In addition, a counter can be tested against an integer constant. These tests are called *standard tests*. An NCM is *reversal-bounded* if each counter is reversal-bounded (i.e., it changes mode between nondecreasing and nonincreasing for some bounded number of times).

A tuple of integers can be encoded as a string by concatenating the unary representations of each integer with a separator. In this way, the binary reachability $\rightsquigarrow_{\mathcal{A}}$ can be treated as a language $\{[\alpha]\#[\beta] : \alpha \rightsquigarrow_{\mathcal{A}} \beta\}$, where $[\alpha]$ and $[\beta]$ are string encodings of configurations α and β separated by a delimiter "#". Obviously, same encoding applies to $\rightsquigarrow_M$, the binary reachability of an NCM M.

When an NCM is reversal-bounded, the emptiness problem is decidable. In fact, we have a stronger characterization.

Theorem 5. *A set of n-tuples of integers is definable by a Presburger formula iff it can be accepted by a reversal-bounded deterministic multicounter machine [19].*

The machines defined above, when used as language recognizers, have a one-way input tape. Suppose a two-way input is used instead. Let $\mathrm{2DCM}(c, r)$ denote the class of deterministic machines with a two-way input tape and c r-reversal-bounded counters. Then the emptiness problem for $\mathrm{2DCM}(c, r)$ when $c \geq 2$ and $r \geq 1$ is undecidable [19]. An interesting special case is when $c = 1$, i.e., there is only one counter. A language is *2DCM-recognizable* if it can be accepted by a $\mathrm{2DCM}(1, r)$.

Theorem 6. *The emptiness problem for 2DCM-recognizable languages is decidable [20].*

It is still open whether Theorem 6 holds for nondeterministic machines. That is, whether the emptiness problem for 2NCM(1,r), which is a nondeterministic r-reversal-bounded one counter machine with a two-way input tape, is decidable.

Given a generalized discrete timed automaton $\mathcal{A}$, consider a subset of configuration pairs, $R_{\mathcal{A}} \subseteq S \times (N^+)^{|\boldsymbol{C}|} \times (N^+)^{|\boldsymbol{X}|} \times S \times (N^+)^{|\boldsymbol{C}|} \times (N^+)^{|\boldsymbol{X}|}$. One can look at $R_{\mathcal{A}}$ as some sort of reachability relation. For a given $\mathcal{A}$, if a 2DCM$(1, r)$ $M_{\mathcal{A}}$ can be effectively constructed such that for every w, w is in $R_{\mathcal{A}}$ iff there exists a w' such that $M_{\mathcal{A}}$ accepts $w\#w'$, then we say that $R_{\mathcal{A}}$ has a *2DCM-padding*. From Theorem 6, it is routine to show the following lemma.

Lemma 1. *(1). The emptiness problem for $\{R_{\mathcal{A}}\}_{\mathcal{A}}$ having 2DCM-paddings is decidable. (2). If $R^1_{\mathcal{A}}$ and $R^2_{\mathcal{A}}$ have 2DCM-paddings, then so do the join $R^1_{\mathcal{A}} \cup R^2_{\mathcal{A}}$ and the composition $R^1_{\mathcal{A}} \circ R^2_{\mathcal{A}}$.*

3 Main Results

Denote $\leadsto^0_{\mathcal{A}}$ to be the binary reachability of a generalized discrete timed automaton $\mathcal{A}$ through a path without clock resets. Note that $\mathcal{A}$ itself can be considered as a nondeterministic multicounter machine. However, tests in $\mathcal{A}$, which are linear relations on clocks and parameterized constants, are not standard. Assignments in $\mathcal{A}$ in the form of $x := x + d$ with d a parameterized constant are also not standard. But, if $\mathcal{A}$ has only unit durations, the assignments are standard except when a clock reset occurs, i.e., $x := 0$. The following result follows our recent results on strong-reversal-bounded NCMs [21].

Lemma 2. *Suppose $\mathcal{A}$ is a generalized discrete timed automaton with unit durations. Then $\leadsto^0_{\mathcal{A}}$ is Presburger.*

The following theorem gives a characterization of the three approximations of $\leadsto_{\mathcal{A}}$ when $\mathcal{A}$ has unit durations. The proof cuts a reachability path of $\mathcal{A}$ into a finite number of phases and each phase can be further characterized by $\leadsto^0_{\mathcal{A}}$ using Lemma 2.

Theorem 7. *Suppose $\mathcal{A}$ is a generalized discrete timed automaton with unit durations. Then $\leadsto^r_{\mathcal{A}}$, $\leadsto^B_{\mathcal{A}}$ and $\leadsto^{\langle B,r\rangle}_{\mathcal{A}}$ are Presburger.*

The case for $\mathcal{A}$ with parameterized durations is more complicated. In principle, $\mathcal{A}$ with parameterized durations can be simulated by an $\mathcal{A}'$ with unit durations. This is done by simply introducing a new clock z to test whether the parameterized duration of a transition is reached, and after the transition is fired z is reset to 0. However, the three approximations on $\mathcal{A}$ are not equivalent to those on $\mathcal{A}'$. The reason is as follows. Consider $\leadsto^0_{\mathcal{A}}$, the 0-reset-bounded approximation of $\mathcal{A}$. $\alpha \leadsto^0_{\mathcal{A}} \beta$ if α can reach β through a path without clock resets. But for $\mathcal{A}'$ each transition with a parameterized duration causes clock z to reset. Thus, a path in $\mathcal{A}'$ witnessing $\alpha \leadsto^0_{\mathcal{A}} \beta$ could have an unbounded number of clock resets. Therefore, for $\mathcal{A}$ with parameterized durations, Theorem 7 is not applicable.

In the following, we consider a generalized discrete timed automaton $\mathcal{A}$ (with parameterized durations). Currently, we cannot show a decidable characterization of $\leadsto^0_{\mathcal{A}}$, since we find it is related to the emptiness problem of 2NCM(1,r), which is still open. However, by restricting $\mathcal{A}$ to be deterministic, the following results can be established. The proof uses the fact that the disjunction of the atomic linear relations that appear in all the enabling conditions of $\mathcal{A}$ is equivalent to a union of convex linear relations.

Lemma 3. *$\leadsto^0_{\mathcal{A}}$ has a 2DCM-padding for a deterministic generalized discrete timed automaton $\mathcal{A}$.*

Again, by using Lemma 3 and the idea of cutting a reachability path of $\mathcal{A}$ into phases, the following theorem gives a characterization of the three approximations of $\leadsto_{\mathcal{A}}$.

Theorem 8. *Suppose $\mathcal{A}$ is a deterministic discrete timed automaton. Then, $\leadsto^r_{\mathcal{A}}$, $\leadsto^B_{\mathcal{A}}$ and $\leadsto^{\langle B,r\rangle}_{\mathcal{A}}$ have 2DCM-paddings.*

4 Verification of Safety Properties

Consider P and I, two sets of configurations of a generalized discrete timed automaton $\mathcal{A}$ definable by Presburger formulas. If, starting from a configuration in I, $\mathcal{A}$ can only reach configurations in P, then P is a *safety property* with respect to the initial condition I. The following is an example:

"starting from a configuration satisfying $x_1 - x_2 + x_3 - x_4 > c + d$, $\mathcal{A}$ cannot reach a configuration satisfying $2x_1 + x_2 < c \wedge x_3 - 3x_4 > 2d - c$."

The *safety analysis problem* is to determine whether P is a safety property with respect to the initial condition I. The following two theorems follow from Theorem 7, Theorem 8, and Lemma 1.

Theorem 9. *The safety analysis problem is decidable for generalized discrete timed automata with unit durations and with any one of the following approximations: r-reset-boundedness, B-boundedness and $\langle B, r\rangle$-crossing-boundedness.*

Theorem 10. *The safety analysis problem is decidable for deterministic generalized discrete timed automata with any one of the following approximations: r-reset- boundedness, B-boundedness and $\langle B, r\rangle$-crossing-boundedness.*

Remark: Theorem 10 can be strengthened. The class of languages accepted by deterministic two-way counter machines with one reversal-bounded counter is closed under Boolean operations [20]. It follows that Theorem 10 remains valid even if the sets of configurations P (property) and I (initial condition) are sets accepted by these machines. ∎

It is desirable to consider the decidability of the safety analysis problem for generalized discrete timed automata under some special form but without using the approximations. One such form is parameterized timed automata with

only one parameterized clock. As stated in Theorem 1 (2), the state reachability problem is decidable. However, surprisingly, the safety analysis problem, i.e.,

"Deciding whether P is a safety property with respect to the initial condition I for a parameterized timed automaton with only one parameterized clock, where both P and I are definable by Presburger formulas"

is still open. This problem is closely related to the open problem of the decidability of the emptiness problem for 2NCM(1,r). It is also worthwhile to point out that the characterizations of the binary reachability of deterministic generalized discrete timed automata under the approximations, as shown in Theorem 8, are not necessarily Presburger. In fact, 2DCM(1,r) can accept a class of nonlinear languages [20].

5 A Verification Example

In practice, allowing parameterized clock constraints and durations makes it possible to specify more complex real-time systems. In this section, we take an example specification [23] of the railroad crossing benchmark [18], which is written in ASTRAL [6]. The specification specifies a system consisting of a set of railroad tracks that intersect a street where cars may cross the tracks. A gate is located at the crossing to prevent cars from crossing the tracks when a train is near. A sensor on each track detects the arrival of trains on that track. The critical requirement of the system is that whenever a train is in the crossing the gate must be down, and when no train has been in between the sensors and the crossing for a reasonable amount of time, the gate must be up. The complete ASTRAL specification of the railroad crossing system, which was written by Paul Kolano, can be found at `http://www.cs.ucsb.edu/~dang`.

The ASTRAL specification includes a global specification and two process specifications: one is `Gate` and the other is `Sensor`. A transition system is specified inside each process specification along with local assumptions and local properties. ASTRAL adopts a modularized view of the specified system at the level of correctness proofs: the global properties in the global specification can be verified by using global assumptions and local properties (instead of the actual transition behaviors) of each process instance; local properties of a process instance can be verified by using the local assumptions, the local imported variable clause and the transition system of the instance, without looking at the transition behaviors of the other process instances (A reader need not worry about the possibility of circular proofs. The ASTRAL proof theory is sound, see [8] for details.).

We take the `Sensor` process specification to see how parameterized durations and parameterized clock constraints are used in ASTRAL. `Sensor` reports whether a train is beyond the crossing, which is indicated by a Boolean variable `train_in_R`. The process specification has only two transitions `enter_R` and `exit_R`, which have parameterized durations `enter_dur` and `exit_dur`, respectively. Transition `enter_R` changes `train_in_R` from `FALSE` to `TRUE` with duration `enter_dur`. Transition `exit_R` sets `train_in_R` back to `FALSE` after the slowest train moves out of the crossing. That is,

```
TRANSITION exit_I
  ENTRY           [ TIME : exit_dur ]
                  train_in_R
                & now - Start ( enter_R ) >= RIImin - exit_dur
  EXIT
                  train_in_R = FALSE,
```

where `now` indicates the current time, `Start(enter_R)` is the most recent start time of transition `enter_R`, and `RIImin` is a parameterized constant indicating the time for the slowest train to move out of the crossing. One of the two transitions is nondeterministically chosen to fire as long as the `ENTRY` condition of the chosen transition is satisfied. However, if neither of them is fireable, the process idles: `now` progresses by one time unit and `train_in_R` does not change.

But, according to this semantics, transition `enter_R` must fire immediately whenever transition `exit_R` completes. This is not the intended behavior of `Sensor`. In fact, it is desirable that `enter_R` fires only when a train actually comes. But `enter_R` specified as above only tells what happened (set `train_in_R` to `TRUE`) when a train comes, instead of the time when a train comes. The pattern of a train's arrival is controlled by the environment of the process. In this specification, transition `enter_R` is declared as exported. That is, it must be called by the environment in order to fire. In `Sensor`, the environment is specified by the environment clause, which states that two consecutive calls of `enter_R` must be separated by at least `RIImin` many time units. The safety property of the process is specified as a *schedule*, which has two conjuncts. The first conjunct of the schedule says that a train will be sensed within `enter_dur` many time units after a call of transition `enter_R` is placed. The second conjunct of the schedule says that the sensor will be reset when the slowest train is beyond the crossing. It is assumed that initially `now` is 0 and `train_in_R` is `FALSE`.

We manually translated `Sensor` into a generalized discrete timed automaton and computed the transitive closure of the one-step transition of the automaton using the Omega Library [26], which is a tool to manipulate Presburger formulas. Experiments were run on a Sun workstation with 4 CPUs and 256M real memory and 512M swap memory. Unfortunately, the closure, even when the durations (`enter_dur` and `exit_ dur`) were set to 1, could not be computed. Then, we used the B-bounded approximation on the automaton with $B = 3$. This time, the binary reachability $\rightsquigarrow_{\mathcal{A}}^{B}$ can be computed in about one minute of CPU time and using 170M memory. But the other two approximation approaches were still too expensive to calculate.

6 Conclusions and Future Work

We studied generalized discrete timed automata with general linear relations over clocks and parameterized constants as clock constraints and with parameterized durations. We focused on three approximation techniques and automata-theoretic characterizations of binary reachability under these approximations. The characterizations allow us to show that the safety analysis problem is decidable with respect to generalized discrete timed automata with unit durations,

and deterministic generalized discrete timed automata with parameterized durations (modulo the approximations). We used an example specification written in ASTRAL to run a number of experiments using one of the approximation techniques. The results of the experiments show that further improvements to the approximations have to be developed, since currently they are not practical for large specifications.

For future work, we want to investigate how the approximation techniques proposed in this paper can be combined with existing image-approximation techniques [12] in debugging infinite state systems. Solutions to this problem would lead to an implementation of an effective specification debugger for large real-time specifications. Another research issue is how to extend the results in this paper to the case of generalized timed automata with dense clocks. Recent ideas used in [9] may provide some insights.

References

1. R. Alur, *"Timed automata"*, *CAV'99*, LNCS **1633**, pp. 8-22
2. R. Alur, C. Courcoubetis and D. Dill, *"Model-checking in dense real time,"* *Information and Computation*, **104** (1993) 2-34
3. R. Alur and D. Dill, *"A theory of timed automata,"* *TCS*, **126** (1994) 183-236
4. R. Alur, T. A. Henzinger, *"A really temporal logic,"* *J. ACM*, **41** (1994) 181-204
5. R. Alur, T. A. Henzinger and M. Y. Vardi, *"Parametric real-time reasoning,"* *STOC'93*, pp. 592-601
6. A. Coen-Porisini, C. Ghezzi and R. A. Kemmerer, *"Specification of real-time systems using ASTRAL,"* *TSE*, **23** (1997) 572-598
7. H. Comon and Y. Jurski, *"Timed automata and the theory of real numbers,"* *CONCUR'99*, LNCS 1664, pp. 242-257
8. A. Coen-Porisini, R. A. Kemmerer and D. Mandrioli, *"A formal framework for ASTRAL intralevel proof obligations,"* *TSE*, **20** (1994) 548-561
9. Z. Dang, *"A decidable binary reachability characterization of timed pushdown automata with dense clocks,"* To appear in *CAV'01*, LNCS
10. Z. Dang and R. A. Kemmerer, *"Using the ASTRAL model checker to analyze Mobile IP,"* *ICSE'99*, pp. 132-141
11. Z. Dang and R. A. Kemmerer, *"A symbolic model checker for testing ASTRAL real-time specifications,"* *RTCSA'99*, pp. 174-181
12. Z. Dang and R. A. Kemmerer, *"Using the ASTRAL symbolic model checker as a specification debugger: three approximation techniques,"* *ICSE'00*, pp. 345-354
13. Z. Dang, O. H. Ibarra and R. A. Kemmerer, `http://www.eecs.wsu.edu/~zdang`, the full version of this paper
14. Z. Dang, P. San Pietro and R. A. Kemmerer, *"On Presburger liveness of discrete timed automata,"* *STACS'01*, LNCS 2010, pp. 132-143
15. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer and J. Su, *"Binary reachability analysis of discrete pushdown timed automata,"* *CAV'00*, LNCS 1855, pp. 69-84
16. T. A. Henzinger and Pei-Hsin Ho, *"HyTech: the Cornell hybrid technology tool,"* *Hybrid Systems II*, LNCS **999**, 1995, pp. 265-294
17. T. A. Henzinger, X. Nicollin, J. Sifakis and S. Yovine. *"Symbolic model checking for real-time systems,"* *Information and Computation*, **111** (1994) 193-244

18. C. Heitmeyer and N. Lynch, *"The generalized railroad crossing: a case study in formal verification of real-time systems," RTSS'94*, pp. 120-131
19. O. H. Ibarra, *"Reversal-bounded multicounter machines and their decision problems," J. ACM*, **25** (1978) 116-133
20. O. H. Ibarra, T. Jiang, N. Tran and H. Wang, "New decidability results concerning two-way counter machines," *SIAM J. Comput.,* **24** (1995) 123-137
21. O. H. Ibarra, J. Su, Z. Dang, T. Bultan and R. A. Kemmerer, *"Counter machines: decidable properties and applications to verification problems,"*, *MFCS'00*, LNCS **1893**, pp. 426-435
22. P. Kolano, *"Tools and techniques for the design and systematic analysis of real-Time systems,"* Ph.D. Thesis, University of California, Santa Barbara, 1999
23. P. Z. Kolano, Z. Dang and R. A. Kemmerer, *"The design and analysis of real-time systems using the ASTRAL software development environment," Annals of Software Engineering*, **7** (1999) 177-210
24. K. G. Larsen, P. Pettersson and W. Yi, *"UPPAAL in a nutshell," International Journal on Software Tools for Technology Transfer*, **1** (1997) 134-152
25. F. Laroussinie, K. G. Larsen and C. Weise, *"From timed automata to logic - and back," MFCS'95*, LNCS **969**, pp. 529-539
26. W. Pugh, *"The Omega test: a fast and practical integer programming algorithm for dependence analysis," CACM*, **8** (1992) 102-104
27. J. Raskin and P. Schobben, *"State clock logic: a decidable real-time logic," HART'97*, LNCS 1201, pp. 33-47
28. T. Wilke, *"Specifying timed state sequences in powerful decidable logics and timed automata,"* LNCS 863, pp. 694-715, 1994
29. S. Yovine, *"A verification tool for real-time systems," International Journal on Software Tools for Technology Transfer*, **1** (1997): 123-133

Multiplicative Adaptive Algorithms for User Preference Retrieval

Zhixiang Chen

Department of Computer science,
University of Texas-Pan American, Edinburg TX 78539, USA,
chen@cs.panam.edu,
http://www.cs.panam.edu/chen/

Abstract. In contrast to the adoption of linear additive query updating techniques in existing popular algorithms for user preference retrieval, in this paper we design two types of algorithms, the multiplicative adaptive query expansion algorithm MA and the multiplicative adaptive gradient search algorithm MG, both of which use multiplicative query expansion strategies to adaptively improve the query vector. We prove that algorithm MA has a substantially better mistake bound than the Rocchio's and the Perceptron algorithms in learning a user preference relation determined by a linear classifier with a small number of non-zero coefficients over the real-valued vector space $[0,1]^n$. We also show that algorithm MG boosts the usefulness of an index term *exponentially*, while the gradient descent procedure does so *linearly*. Our work also generalize the algorithm Winnow in the following aspects: various updating functions may be used; multiplicative updating for a weight is dependent on the value of the corresponding index term, which is more realistic and applicable to real-valued vector space; and finally, a number of documents which may or may not be counterexamples to the algorithm's current classification are allowed. Practical implementations of algorithms MA and MG have been underway in the next stage development of our intelligent web search tools.

1 Vector Space and User Preference

Let $\mathcal{R}$ be the set of all real values, and let $\mathcal{R}^+$ be the set of all non-negative real values. Let n be a positive integer. In the vector space model in information retrieval [11], a collection of n indexing terms $T_1, T_2, \ldots, T_n$ are used to represent documents and queries. Each document $\mathbf{d}$ is represented as a vector $\mathbf{d} = (d_1, \ldots, d_n)$ such that for any i, $1 \leq i \leq n$, the i-th component of $\mathbf{d}$ is used to determine the relevance (or weight) of the i-th term T_i in the document. Because a document vector can be normalized, without loss of generality we only consider the real-valued vector space $[0,1]^n$ in this paper. Given any two vectors $\mathbf{x} = (x_1, \ldots, x_n), \mathbf{y} = (y_1, \ldots, y_n) \in [0,1]^n$ (or $\mathcal{R}^n$), we use $\mathbf{x} \bullet \mathbf{y}$ to denote their inner product $x_1y_1 + \cdots + x_ny_n$.

Let $\mathcal{D}$ be a collection of documents. Given any two documents in $\mathcal{D}$, we assume that a user would *prefer* one to other or regard both as being *equivalent*

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 540–549, 2001.
© Springer-Verlag Berlin Heidelberg 2001

with respect to her information needs (or search queries). In other words, user preference of documents in $\mathcal{D}$ defines a preference relation $\prec$ over $\mathcal{D}$ as follows:

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \prec \mathbf{d}' \Longleftrightarrow \textit{ the user } \textit{prefers } \mathbf{d}' \textit{ to } \mathbf{d}.$$

It has been shown in [1] that if a user preference relation $\prec$ is a *weak order* satisfying some additional conditions then it can be represented by a linear classifier. That is, there is a query vector $\mathbf{q} = (q_1, \ldots, q_n) \in \mathcal{R}^n$ such that

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \prec \mathbf{d}' \Longleftrightarrow \mathbf{q} \bullet \mathbf{d} < \mathbf{q} \bullet \mathbf{d}'. \tag{1}$$

In general a linear classifier over the vector space $[0, 1]^n$ is a pair of $(\mathbf{q}, \theta)$ which classifies any document $\mathbf{d}$ as relevant if $\mathbf{q} \bullet \mathbf{d} > \theta$, or irrelevant otherwise, where the query vector $\mathbf{q} \in \mathcal{R}^n$ and the classification threshold $\theta \in \mathcal{R}^+$. Recall that $\mathbf{q} \bullet \mathbf{d}$ is usually used as the relevance rank or (score) of the document d with respect to user preference.

A natural way to understand a user preference relation $\prec$ is *document ranking*: A user prefers a document $\mathbf{d}$ to a document $\mathbf{d}'$, if and only if she ranks $\mathbf{d}$ higher than $\mathbf{d}'$. When a user has no preference of $\mathbf{d}$ to $\mathbf{d}'$ nor $\mathbf{d}'$ to $\mathbf{d}$, then she is really not interested in how those two documents are actually ranked. Based on such understanding, the following linear *acceptable ranking strategy* was proposed in [13]:

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \prec \mathbf{d}' \Longrightarrow \mathbf{q} \bullet \mathbf{d} < \mathbf{q} \bullet \mathbf{d}', \tag{2}$$

where $\mathbf{q} \in R^n$ is the query vector determined by the user.

Let $\mathcal{D}_r$ be the set of all relevant documents in $\mathcal{D}$ with respect to a user's information needs (or search queries), and $\mathcal{D}_{ir}$ the set of all irrelevant documents. If we assume that a user preference relation has a simple structure with only two levels, i.e., one level consisting of all relevant documents and the other consisting of all irrelevant documents. Within the same level, no preference is made between any two documents. Then, finding a user preference relation satisfying the expression (1) is equivalent to the problem of finding a linear classifier $(\mathbf{q}, \theta)$ over $[0, 1]^n$ with the property

$$\forall \mathbf{d} \in \mathcal{D}, \quad \mathbf{d} \in \mathcal{R} \Longleftrightarrow \mathbf{q} \bullet \mathbf{d} > \theta, \tag{3}$$

where $\mathbf{q} \in \mathcal{R}^n$ is the query (or weight) vector. Similarly, finding a linear acceptable ranking strategy satisfying expression (2) is equivalent to the problem of finding a query vector $\mathbf{q} \in R^n$ with the property

$$\forall \mathbf{d}, \mathbf{d}' \in \mathcal{D}, \quad \mathbf{d} \in \mathcal{D}_{ir} \textit{ and } \mathbf{d}' \in \mathcal{D}_r \Longrightarrow \mathbf{q} \bullet \mathbf{d} < \mathbf{q} \bullet \mathbf{d}'. \tag{4}$$

The goal of relevance feedback in information retrieval is to identify a user preference relation $\prec$ with respect to her information needs from documents judged by that user. Since user preference relations vary from different users and may have various unknown representations, it is not easy for an information system to find such relations. The existing popular relevance relevance algorithms

basically use linear additive query expansion methods to find a user preference relation as follows:

- Start with an initial query vector $\mathbf{q}_0$.
- At any step $k \geq 0$, improve the k-th query vector $\mathbf{q}_k$ to

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \alpha_1 \mathbf{d}_1 + \cdots + \alpha_s \mathbf{d}_s, \tag{5}$$

where $\mathbf{d}_1, \ldots, \mathbf{d}_s$ are the documents judged by user, and the updating factors $\alpha_i \in \mathcal{R}$ for $i = 1, \ldots, s$.

One particular and well-known example of relevance feedback is Rocchio's similarity-based relevance feedback [7]. Depending on how updating factors are used in improving the k-th query vector as in expression (5), a variety of relevance feedback algorithms have been designed [11]. A similarity-based relevance feedback algorithm is essentially an adaptive supervised learning algorithm from examples [12]. The goal of the algorithm is to learn some unknown classifier (such as the linear classifier in expression (3)) that is determined by a user's information needs to classify documents as relevant or irrelevant. The learning is performed by modifying or updating the query vector that serves as the hypothetical representation of the collection of all relevant documents. The technique for updating the query vector is linear additions of the vectors of documents judged by the user. This type of linear additive query updating technique is similar to what used by the Perceptron algorithm [10]. The linear additive query updating technique has some disadvantage: It's *converging rate* to the unknown target classifier is *slow*, because it has been proved that the Perceptron algorithm [8] and the Rocchio's relevance feedback algorithm [5] with any of the four typical similar measures [11] have an $\Omega(n)$ lower bound on their performance of learning in the n dimensional binary vector space. In the real world of web search, a huge number of terms (usually, keywords) are used to index web documents. To make the things even worse, no users will have the patience to try, say, more than 10 iterations of relevance feedback in order to gain some significant search precision increase. This implies that the traditional linear additive query updating method may be too slow to be applicable to web search, and this motivates us to design new and *faster* query updating methods for user preference retrieval in section 4.

For a user preference with respect to her information needs, for any index term T, define

- $|\mathcal{D}|$ = the total number of documents in the collection $\mathcal{D}$
- $|\mathcal{D}_r|$ = the total number of relevant documents
- η = the number of documents indexed by T
- γ = the number of relevant documents indexed by T

A gradient descent procedure has been designed in [13] to find an acceptable ranking strategy satisfying expression (2). The idea of the procedure is to minimize ranking errors through linear additions of $\mathbf{d}' - \mathbf{d}$ for all pairs of documents $\mathbf{d}'$ and $\mathbf{d}$ that, according to expression (3), are ranked incorrectly. This strategy is similar to what used by the Perceptron algorithm [6]. When the gradient descent procedure is applied to find an acceptable ranking satisfying expression

(4), it has been shown [13] that after the first iteration from a zero initial query vector the procedure weighs an index term T *linearly* in $\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|}$, an approximate measure of the usefulness of index term T for distinguishing relevant and irrelevant documents. It has also shown that under certain *good* index term probability distribution the above usefulness measure for the index term T reaches its expected maximum when $\eta = 0.5|\mathcal{D}|$, a justification for choosing mid-frequency terms in indexing [11]. In contrast to the linear additive query updating strategy used in the gradient descent procedure [13] for minimizing ranking errors, in section 4 we will design a new algorithm with the multiplicative query updating strategy. We will show that after the first iteration from a zero initial query vector our new algorithm weighs an index term T *exponentially* in $\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|}$. This means that *exponentially large gaps* will be generated for index terms with respect to measures of their usefulness. Hence, a document with a *good* index term will be ranked *exponentially higher* than one without the *good* index term, thus more ideal ranking effect will be generated for users.

2 The Multiplicative Adaptive Query Expansion Algorithm

In this section we design a multiplicative query updating technique to identify a user preference relation satisfying expression (1). We believe that linear additive query updating yields some *mild* improvement on the hypothetical query vector towards the target user preference. We want a query updating technique that can yield *dramatic* improvements so that the hypothetical query vector can be moved towards the target in a much faster pace. Our idea is that when an index term is judged by the user, its corresponding value in the hypothetical query vector should be boosted by a multiplicative factor that is dependent on the value of the term itself.

Algorithm $MA(q_0, f, \theta)$:
(i) Inputs:
$\mathbf{q}_0$*, the non-negative initial query vectory*
$f(x) : [0, 1] \longrightarrow R^+$*, the updating function*
$\theta \geq 0$*, the classification threshold*
(ii) Set $k = 0$.
(iii) Classify and rank documents with the linear classifier $(\mathbf{q}_k, \theta)$.
(iv) While (the user judged the relevance of a document $\mathbf{d}$*) do* {
for $i = 1, \ldots, n$*, do* {
/* $\mathbf{q}_k = (q_{1,k}, \ldots, q_{n,k})$, $\mathbf{d} = (d_1, \ldots, d_n)$ */
if $(\mathbf{d}_i \neq 0)$ {
if $(\mathbf{q}_k \neq 0)$ *set* $q_{i,k+1} = q_{i,k}$ *else set* $q_{i,k+1} = 1$ /* *adjustment* */
if ($\mathbf{d}$ *is relevant)* /* *promotion* */
set $q_{i,k+1} = (1 + f(d_i))q_{i,k+1}$
else /* *demotion* */

set $q_{i,k+1} = \frac{q_{i,k+1}}{1+f(d_i)}$

}

}

}

(v) If the user has not judged any document in the k-th step, then stop. Otherwise, let $k = k + 1$ *and go to step (iv).*

/ The end of the algorithm MA */*

In this paper, we are only interested in non-decreasing updating functions $f(x) : [0, 1] \Longrightarrow \mathbf{R}^+$, because we want multiplicative updating for an index term is proportional to the value of the term.

We now analyze the the performance of algorithm MA when it is used to identify a user preference satisfying expression (3), a linear classifier $(\mathbf{q}, 0)$. Here, we consider a zero threshold. we say that algorithm makes an classification error at step k if the user judged a document as a counterexample to the algorithm's current hypothesis. We estimate the total number of classification errors algorithm MA will make based on the worst-case analysis. We also let at most one counterexample may be provided to the algorithm at each step. From now on to the end of this section we assume that $\mathbf{q}$ is a non-negative query vector with m non-zero components and $\theta > 0$. Define

$$\beta = min\{q_1, \ldots, q_n\}.$$

Definition 1. *We say that documents in the collection* $\mathcal{D}$ *are indexing with respect to a threshold* δ*,* $0 < \delta \leq 1$*, if for any document* $\mathbf{d} = (d_1, \ldots, d_n) \in \mathcal{D}$ *we have either* $d_i = 0$ *or* $\delta \leq d_i$*,* $1 \leq i \leq n$*.*

In other words, when a document is indexed with respect to a threshold δ, any index term with a value below the threshold δ is considered not significant, and hence set to zero. Recall that in the vector space a document and its vector have the equivalent meaning, so we may not distinguish the two concepts.

Lemma 1. *Assume that documents are indexed with respect to a threshold* δ*. Let* u *denote the total number of promotions algorithm MA needs to find the linear classifier* $(\mathbf{q}, 0)$*. Let* m *denote the number of non-zero components in* $\mathbf{q}$*. Then,*

$$u \leq \frac{m \log \frac{\theta}{\beta\delta}}{\log(1 + f(\delta))}.$$

Proof. Without loss of generality, we may further assume that the m non-zero components of $\mathbf{q}$ are $q_1, \ldots, q_m$. When a promotion occurs at step k, a relevant document $\mathbf{d}$ is given to algorithm as a counterexample its classification. Because of document indexing with respect to the threshold δ, there is some i with $1 \leq i \leq m$ such that $d_i \geq \delta$. This means that the i-th component $q_{i,k}$ of the query vector $\mathbf{q}_k$ will be promoted to

$$q_{i,k+1} = (1 + f(d_i))q_{i,k} > (1 + f(\delta))q_{i,k}, \tag{6}$$

because f is non-decreasing. Since $q_{i,k}$ will never be demoted, it follows from expression (6) that $q_{i,k}$ can be promoted at most

$$\frac{\log \frac{\theta}{\beta\delta}}{\log(1+f(\delta))} \tag{7}$$

times. Since each promotion yields a promotion for at least one $q_{i,k}$ for $1 \le i \le m$, the total number of promotions u is at most m times the value given in expression (7).

Theorem 1. *Assume that documents are indexed with respect to a threshold δ. Let T denote the total number of classification errors algorithm MA makes in order to find the linear classifier $(\mathbf{q}, 0)$ over the real-valued vector space $[0,1]^n$. Let m denote the number of non-zero components in $\mathbf{q}$. Then,*

$$T \le \frac{[(1+f(1))(n-m)+\gamma](1+f(\delta))\delta}{f(\delta)\theta} + \left(\frac{f(1)(1+f(\delta))}{f(\delta)} + 1\right)\frac{m \log \frac{\theta}{\beta\delta}}{\log(1+f(\delta))}$$

(Hence, if $\theta = \frac{n}{k}$ is chosen, $T = O(k \log n)$.)

Proof. Without loss of generality, we may assume that the m non-zero components of $\mathbf{q}$ are $q_1, \ldots, q_m$. We estimate the sum of the weights $\sum_{i=1}^{n} q_{i,k}$. Let u and v be the number of promotion steps and the number of demotion steps occurred during the learning process, respectively. Let t_k denote the number of zero components in $\mathbf{q}_k$ at step k. Note that once a component of $\mathbf{q}_k$ is promoted to a non-zero value, it will never become zero again. For a promotion at step k with respect to a relevant document $\mathbf{d}$ judged by the user, for $i = 1, \ldots, n$, we have

$$q_{i,k+1} = \begin{cases} q_{i,k}, & if\ d_i = 0, \\ (1+f(d_i)), & if\ d_i \neq 0\ and\ q_{i,k} = 0, \\ (1+f(d_i))q_{i,k}, & if\ d_i \neq 0\ and\ q_{i,k} \neq 0. \end{cases}$$

Since a promotion only occurs when

$$\mathbf{q}_k \bullet \mathbf{d} = \sum_{i=1}^{n} d_i q_{i,k} = \sum_{d_i \neq 0\ and\ q_{i,k} \neq 0} q_{i,k} < \theta,$$

we have

$$\begin{aligned} \sum_{i=1}^{n} q_{i,k+1} &= \sum_{d_i \neq 0\ and\ q_{i,k}=0} q_{i,k+1} + \sum_{d_i \neq 0\ and\ q_{i,k} \neq 0} q_{i,k+1} + \sum_{d_i = 0} q_{i,k+1} \\ &= \sum_{d_i \neq 0\ and\ q_{i,k}=0} (1+f(d_i)) + \sum_{d_i \neq 0\ and\ q_{i,k} \neq 0} (1+f(d_i))q_{i,k} + \sum_{x_i = 0} q_{i,k} \\ &\le (1+f(1))t_k + \frac{1+f(1)}{\delta} \sum_{d_i \neq 0\ and\ q_{i,k} \neq 0} \delta q_{i,k} + \sum_{i=1}^{n} q_{i,k}. \end{aligned}$$

$$\leq (1+f(1))t_k + \frac{1+f(1)}{\delta} \sum_{d_i \neq 0 \ and \ q_{i,k} \neq 0} d_i q_{i,k} + \sum_{i=1}^{n} q_{i,k}.$$

$$\leq (1+f(1))t_k + \frac{1+f(1)}{\delta}\theta + \sum_{i=1}^{n} q_{i,k}. \tag{8}$$

For a demotion at step k with respect to an irrelevant document $\mathbf{d}$ judged by the user, for $i = 1, \ldots, n$, we have

$$q_{i,k+1} = q_{i,k} - (1 - \frac{1}{1+f(d_i)})q_{i,k} \leq q_{i,k} - (1 - \frac{1}{1+f(\delta)})q_{i,k}.$$

Since a demotion occurs only when $\sum_{i=1}^{n} d_i q_{i,k} > \theta$, we have

$$\begin{aligned}
\sum_{i=1}^{n} q_{i,k+1} &\leq \sum_{i=1}^{n} q_{i,k} - (1 - \frac{1}{1+f(\delta)}) \sum_{i=1}^{n} q_{i,k} \\
&= \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1+f(\delta))\delta} \sum_{i=1}^{n} \delta q_{i,k} \\
&\leq \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1+f(\delta))\delta} \sum_{i=1}^{n} d_i q_{i,k} \\
&\leq \sum_{i=1}^{n} q_{i,k} - \frac{f(\delta)}{(1+f(\delta))\delta}\theta
\end{aligned} \tag{9}$$

Let the sum of the initial weights be σ. Hence, after u promotions and v demotions,

$$\begin{aligned}
\sum_{i=1}^{n} q_{i,k+1} &\leq (1+f(1)) \sum^{u} t_s + \sum_{i=1}^{n} q_{i,0} + \frac{f(1)\theta u}{\delta} - \frac{f(\delta)\theta v}{(1+f(\delta))\delta} \\
&\leq (1+f(1))(n-m) + \sigma + \frac{f(1)\theta u}{\delta} - \frac{f(\delta)\theta v}{(1+f(\delta))\delta}
\end{aligned}$$

Note that at any step the weights are never negative. It follows from the above relation that

$$v \leq \frac{[(1+f(1))(n-m)+\sigma](1+f(\delta))\delta}{f(\delta)\theta} + \frac{f(1)(1+f(\delta))u}{f(\delta)}. \tag{10}$$

It follows from Lemma 3.2 and the above analysis that the total number of promotions and demotions, i.e., the total number of classification errors T, is bounded by

$$\begin{aligned}
T &\leq v + u \\
&\leq \frac{[(1+f(1))(n-m)+\sigma](1+f(\delta))\delta}{f(\delta)\theta} + \frac{f(1)(1+f(\delta))u}{f(\delta)} + u \\
&\leq \frac{[(1+f(1))(n-m)+\sigma](1+f(\delta))\delta}{f(\delta)\theta} + (\frac{f(1)(1+f(\delta))}{f(\delta)} + 1)\frac{m \log \frac{\theta}{\beta\delta}}{\log(1+f(\delta))}
\end{aligned}$$

This completes our proof.

3 The Multiplicative Gradient Search Algorithm

In this section, we design algorithm MG for finding a query vector $\mathbf{q}$ satisfying the acceptable ranking strategy condition (2). Algorithm MG uses a multiplicative query updating technique to minimize its ranking errors.

Algorithm $MG(q_0, f)$:
(i) Inputs:
$\mathbf{q}_0$*, the non-negative initial query vectory*
$f(x) : [0,1] \longrightarrow R^+$*, the update function*
(ii) Set $k = 0$.
(iii) Let $\mathbf{q}_k$ *be the query vector at step* k*. Identify the set of mistakes*

$$\Gamma(\mathbf{q}_k) = \{< \mathbf{d}, \mathbf{d}' > | d \prec d', \mathbf{q}_k \bullet \mathbf{d} \geq \mathbf{q}_k \bullet \mathbf{d}'\}.$$

If $\Gamma(\mathbf{q}_k) = \emptyset$*, stop.*
(iv) For each pair $< \mathbf{d}, \mathbf{d}' > \in \Gamma(\mathbf{q}_k)$*, do* {
for $i = 1, \ldots, n$*, do* {
if $(\mathbf{d}'_i \neq 0)$ {
if $(\mathbf{q}_{i,k} \neq 0)$ *set* $q_{i,k+1} = q_{i,k}$ *else set* $q_{i,k+1} = 1$ /* *adjustment* */
set $q_{i,k+1} = (1 + f(d'_i))q_{i,k+1}$ /* *promotion* */
}
if $(\mathbf{d}_i \neq 0)$
set $q_{i,k+1} = \frac{q_{i,k+1}}{1+f(d_i)}$ /* *demotion* */
}
}
(v) Let $k = k + 1$ *and go to step (iv).*
/* *The end of the algorithm MG* */

Theorem 2. *Assume that algorithm MG is applied to find an acceptable ranking strategy satisfying condition (4). If one chooses the initial query vector* $\mathbf{q}_0 = \mathbf{0}$*, then after the first iteration, for any* $1 \leq i \leq n$*, the weight* $q_{i,1}$ *for the* i*-th indexing term in* $\mathbf{q}_1$ *is*

$$q_{i,1} = \prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} \frac{1 + f(d'_i)}{1 + f(d_i)}.$$

In particular, when a linear updating function $f(x) = \alpha$ *is chosen,*

$$q_{i,1} = (1 + \alpha)^{|\mathcal{D}| \cdot |\mathcal{D}_r| (\frac{\eta}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|})}.$$

Proof. Since the acceptable ranking strategy satisfying condition (4), it follows from $\mathbf{q}_0 = \mathbf{0}$ that at the first iteration one obtains

$$\Gamma(\mathbf{q}_0) = \{< \mathbf{d}, \mathbf{d}' > | \mathbf{d} \in \mathcal{D}_{ir}, \mathbf{d}' \in \mathcal{D}_r\}.$$

Note that during the first iteration, for each pair $< \mathbf{d}, \mathbf{d}' > \in \Gamma(\mathbf{q}_0)$, a promotion performed for $\mathbf{d}'$ and a demotion is performed for $\mathbf{d}$. This implies that for any

i, $1 \leq i \leq n$, after the first iteration the value $q_{i,1}$ with respect to the i-th index term is

$$q_{i,1} = \prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} \frac{1 + f(d'_i)}{1 + f(d_i)}.$$

When f is a constant function mapping with $f(x) = \alpha$, it easily follows from the above expression

$$\begin{aligned} q_{i,1} &= \prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} \frac{1 + f(d'_i)}{1 + f(d_i)} = \frac{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} (1 + f(d'_i))}{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} (1 + f(d_i))} \\ &= \frac{\prod_{\mathbf{d}' \in \mathcal{D}_r} \prod_{\mathbf{d} \in \mathcal{D}_{ir}} (1 + f(d'_i))}{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} \prod_{\mathbf{d}' \in \mathcal{D}_r} (1 + f(d_i))} = \frac{\prod_{\mathbf{d}' \in \mathcal{D}_r} (1 + f(d'_i))^{|\mathcal{D}_{ir}|}}{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} (1 + f(d_i))^{|\mathcal{D}_r|}} \\ &= \frac{\prod_{\mathbf{d}' \in \mathcal{D}_r} (1 + \alpha)^{|\mathcal{D}_{ir}|}}{\prod_{\mathbf{d} \in \mathcal{D}_{ir}} (1 + \alpha)^{|\mathcal{D}_r|}} = \frac{(1 + \alpha)^{|\mathcal{D}_{ir}|\gamma}}{(1 + \alpha)^{|\mathcal{D}_r|(\eta - \gamma)}} \\ &= (1 + \alpha)^{|\mathcal{D}_{ir}|\gamma - |\mathcal{D}_r|(\eta - \gamma)} = (1 + \alpha)^{(|\mathcal{D}| - |\mathcal{D}_r|)\gamma - |\mathcal{D}_r|(\eta - \gamma)} \\ &= (1 + \alpha)^{|\mathcal{D}| \cdot |\mathcal{D}_r| (\frac{\gamma}{|\mathcal{D}_r|} - \frac{\eta}{|\mathcal{D}|})} \end{aligned}$$

This completes our proof.

4 Concluding Remarks

Our motivations of the work in this paper come from the reality of web search: Web search users usually have no patience to try, say, more than 10 iterations of relevance feedback for some intelligent search system in order to gain certain significant search precision increase. Existing popular algorithms for user preference retrieval have their own beauty and advantages. Because of the adoption of *linear additive* query updating techniques, when used to identify user preference determined by a linear classifier those algorithms, such as Rocchio's algorithm and its variants, the Perceptron algorithm and gradient descent procedure, have either a *slow* converging rate or *small* boosting on the usefulness of an index term.

In contrast to the adoption of linear additive query updating techniques in the those existing algorithms, we design two types of algorithms, the multiplicative adaptive query expansion algorithm MA and the multiplicative adaptive gradient search algorithm MG, both of which use multiplicative query updating techniques to adaptively improve the query vector. We have analyzed the worst-case performance of algorithms MA and MG. We plan to investigate their average-case performance. We feel that this task is very challenging. We also plan to conduct empirical studies to understand the behaviors of the algorithm MA and GA with real world data sets. Finally, we would like to point out that implementations of algorithms MA and MG have been underway in the next stage development of our intelligent web search tools [2,3,4].

Acknowledgment

The author thanks Professor Ada Fu for her comments and critiques on our algorithm TW2 and WebSail system [4] which inspired the work in this paper.

References

1. P. Bollmann and S.K.M. Wong. Adaptive linear information retrieval models. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, 1987.
2. Z. Chen and X. Meng. Yarrow: A real-time client site meta search learner. In *Proceedings of the AAAI 2000 Workshop on Artificial Intelligence for Web Search*, pages 12–17, Austin, July 2000.
3. Z. Chen, X. Meng, R.H. Fowler, and B. Zhu. FEATURES: Real-time adaptive feature learning and document learning. Journal of the American Society for Information Science, June 2001.
4. Z. Chen, X. Meng, B. Zhu, and R. Fowler. WebSail: From on-line learning to web search. In *Proceedings of the 2000 International Conference on Web Information Systems Engineering (the full version will appear in Journal of Knowledge and Information Science, the special issue of WISE'00)*, pages 192–199, Hong Kong, June 2000.
5. Z. Chen and B. Zhu. Some formal analysis of the Rocchio's similarity-based relevance feedback algorithm. In D.T. Lee and Shang-Hua Teng, editors, *Proceedings of the Eleventh International Symposium on Algorithms and Computation, Lecture Notes in Computer Science 1969* (the full version will appear in Journal of Information Retrieval), pages 108–119. Springer-Verlag, 2000.
6. R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. John Wiley, 1973.
7. Jr. J.J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The Smart Retrieval System - Experiments in Automatic Document Processing*, pages 313–323, Englewood Cliffs, NJ, 1971. Prentice-Hall, Inc.
8. J. Kivinen, M.K. Warmuth, and P. Auer. The perceptron algorithm vs. winnow: linear vs. logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, pages 325–343, 1997.
9. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
10. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–407, 1958.
11. G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.
12. G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1990.
13. S.K.M. Wong, Y.Y. Yao, and P. Bollmann. Linear structures in information retrieval. In *Proceedings of the 1988 ACM-SIGIR Conference on Information Retrieval*, pages 219–232, 1988.

Parametric Scheduling for Network Constraints

K. Subramani

Department of Computer Science and Electrical Engineering,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

Abstract. The problem of parametric scheduling is concerned with checking whether a job-set is parametrically schedulable, subject to a set of imposed constraints. In real-time scheduling, parameterization of the schedule plays an important role in extending the flexibility of the scheduler, particularly in the presence of variable execution times. It has been shown that the existence of parametric schedules can be determined in polynomial time when the constraints are restricted to those that can be represented by a network, unimodular matrix. In this paper, we extend the class of constraints for which parametric schedules can be determined efficiently to include *network constraints*, such as weighted sum of completion times.

1 Introduction

Uncertainty in problem parameters is often a feature of real-time scheduling [16]. In the literature, there exist two broad approaches to address uncertainty, viz. stochastic and deterministic. In stochastic scheduling, the goal is to provide probabilistic guarantees that the constraints imposed on the job-set will be met, under the assumption that the non-constant problem paramters belong to a fixed well-understood distribution [11,4,14]. This approach is not applicable in the case of "hard" real-time systems [16,12,9], where the guarantees have to be absolute i.e. there is no room for error. Hard real-time systems are typically composed of mission-critical tasks, wherein the consequences of failure, i.e. a violation of constraints can be catastrophic. Hard real-time systems call for deterministic approaches to the issue of uncertainity. Some of the common approaches include *worst-case* assumptions [11], static scheduling [19] and parametric scheduling [5]. Worst-case assumptions regarding execution times (say) run the risk of constraint violation at *run-time* [17] and hence the strategy is not always correct. Static approaches make extremely conservative assumptions about each constraint in order to determine the existence of a feasible schedule. This approach is correct inasmuch as the goal is to provide a set of start-times that cannot cause constraint violation. However, static scheduling is extremely inflexible and even simple constraint sets (see Section §3) may not have static schedules. Parametric scheduling attempts to combine the guarantees provided by static scheduling with the flexibility of stochastic scheduling to enhance the class of constraints for which feasible schedules exist.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 550–560, 2001.
© Springer-Verlag Berlin Heidelberg 2001

In real-time scheduling literature, it has been shown that polynomial time algorithms exist for determining parametric schedules, when the constraints are restricted to be "standard" [5,2]. In this paper we are concerned with the following question: *Can parametric schedulability be determined efficiently for non-standard constraints, with at most* 2 *tasks per constraint ?* We provide an affirmative answer to the above question by designing a polynomial time algorithm for *network constraints*.

The rest of this paper is organized as follows: Section §2 provides a formal description of the problem under consideration. Section §3 discusses the motivation underlying our research while related approaches are detailed in Section §4. Our algorithm and the accompanying analysis are presented in Section §5.

2 Statement of Problem

2.1 Job Model

Assume an infinite time-axis divided into windows of length L, starting at time $t = 0$. These windows are called *periods* or *scheduling windows*. There is a set of non-preemptive, ordered jobs, $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that execute in each scheduling window.

2.2 Constraint Model

The constraints on the jobs are described by System (1):

$$\mathbf{A}.[\boldsymbol{s}, \boldsymbol{e}] \leq \boldsymbol{b}, \quad \boldsymbol{e} \in \mathbf{E}, \tag{1}$$

where,

- $\mathbf{A}$ is an $m \times 2.n$ rational matrix, in which every row represents a *network* constraint (to be defined below),
- $\mathbf{E}$ is the convex set defined by the axis-parallel hyper-rectangle (`aph`)

$$\boldsymbol{\Upsilon} = [l_1, u_1] \times [l_2, u_2] \times \ldots [l_n, u_n] \tag{2}$$

- $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$ is the start time vector of the jobs, and
- $\boldsymbol{e} = [e_1, e_2, \ldots, e_n] \in \mathbf{E}$ is the execution time vector of the jobs

The characterization of $\mathbf{E}$ as an `aph` is intended to model the fact that the execution time e_i of job J_i is not a fixed constant, but can assume any value in the pre-specified range $[l_i, u_i]$, depending on factors such as *loop-length*. In real-time systems such as Maruti, a number of runs of the job-set are carried out to statistically estimate the range $[l_i, u_i]$ [10].

Observe that System (1) can be rewritten in the form

$$\mathbf{G}.\boldsymbol{s} + \mathbf{H}.\boldsymbol{e} \leq \boldsymbol{b}, \quad \boldsymbol{e} \in \mathbf{E} \tag{3}$$

Definition 1. *Standard Constraint: A constraint is defined to be standard, if represents a relative separation relationship between at most two tasks, i.e. matrices* $\mathbf{G}$ *and* $\mathbf{H}$ *are flow graph, unimodular, with the added provision, that the entry* $\mathbf{H}[i,j]$ *must equal* $\mathbf{G}[i,j]$ *if it is non-zero.*

For instance a constraint of the form: $s_i - s_j \leq -8$, which specifies that Job J_j must start 8 units after job J_i starts is a standard constraint. A detailed description of standard constraints is available in [5]. Standard constraints can be represented by edges of a flow graph [18].

Definition 2. *Network Constraint: A constraint is said to be a network constraint, if it can be expressed in the form*

$$a.s_i + b.s_j \leq c.e_i + d.e_j, a, b, c, d \in \mathbf{Q} \tag{4}$$

i.e. all co-efficients are arbitrary rational numbers.

Network constraints also have a graph structure; the "edge" representing the set of constraints between two jobs J_i and J_j form a polyhedron in the 4 variables s_i, e_i, s_j, e_j, with e_i and e_j being universally quantified [6].

2.3 Query Model

Suppose that job J_a has to be dispatched. We assume that the dispatcher has access to the start times $\{s_1, s_2, \ldots, s_{a-1}\}$ and execution times $\{e_1, e_2, \ldots, e_{a-1}\}$ of the jobs $\{J_1, J_2, \ldots, J_{a-1}\}$.

Definition 3. *A parametric schedule of an ordered set of jobs, in a scheduling window, is a vector* $\boldsymbol{s} = [s_1, s_2, \ldots, s_n]$*, where* s_1 *is a rational number and each* $s_i, i \neq 1$ *is a function of the start time and execution time variables of jobs sequenced prior to job* J_i*, i.e.* $\{s_1, e_1, s_2, e_2, \ldots, s_{i-1}, e_{i-1}\}$*. Further, this vector should satisfy the constraint system (1) for all execution time vectors* $\boldsymbol{e} \in \boldsymbol{\Upsilon}$*.*

The combination of the Job model, Constraint model and the Query model constitutes a scheduling problem specification within the `E-T-C` scheduling framework [17].

The Parametric Scheduling problem is concerned with the following two issues:

1. Determining whether the given job-set has a parametric schedule, i.e. a schedule as defined in Definition (3);
2. Computing the start-time of a job in each scheduling window, assuming that
   (a) The parametric schedulability query has been decided affirmatively, and
   (b) The start and execution times of all jobs sequenced before it, are provided.

   This corresponds to the online dispatching phase.

The discussion above directs us to the following formulation of the parametric schedulability query:

$$\exists s_1 \, \forall e_1 \in [l_1, u_1] \, \exists s_2 \, \forall e_2 \in [l_2.u_2], \ldots \exists s_n \, \forall e_n \in [l_n, u_n] \quad \mathbf{A}.[\boldsymbol{s}, \boldsymbol{e}] \leq \boldsymbol{b} \quad ? \tag{5}$$

3 Motivation

The motivation for Parametric Scheduling has been provided in great detail in [5] and [17]. The key issue is that a simple constraint system such as:

$$\begin{aligned} s_1 + e_1 &\leq s_2 \\ s_2 &\leq s_1 + e_1 + 1 \\ e_1 &\in [3, 5] \end{aligned}$$

does not have a static schedule i.e. no assignment of rational numbers to the start times can guarantee the meeting of both constraints for all values of e_1 in the range $[3, 5]$. Note that the schedule represented by $\boldsymbol{s} = [0, s_1 + e_1]^T$ is a valid, feasible, albeit parametric schedule.

An interesting line of research is obtaining schedules that satisfy certain optimization criteria. The complexity of *Parametric Optimization* for general constraints is not known. However, we can approximate optimization functions involving at most two tasks through the use of network constraints. Optimization criteria, formulated as *performance metrics* arise in various situations including Job-shop, Flow-shop and Machine-Shop [11]. Typical performance metrics are Makespan, Sum of Completion times, Weighted sum of completion times (also called aggregate metrics in the Operations Research literature), Lateness and Tardiness. For instance, the need to minimize the weighted sum of completion times of jobs J_1 and J_2 can be approximated through $w_1.(s_1+e_1)+w_2.(s_2+e_2) \leq k$, for suitably chosen k [16]. The other performance metrics can be similarly approximated.

4 Related Work

The concept of Parametric Scheduling was introduced in [12]. In [5], polynomial time algorithms were presented for the case, when the constraints imposed on the job-set are "standard". [18] argued that "standard" constraints could be represented by a flow, graph (i.e. the matrix $\mathbf{A}$ is network, unimodular). [2] extends the "standard constraint" model to include the case, in which constraints can exist between adjacent windows. In [17], it is shown that the problem can be solved in `PSPACE`, when the constraints are arbitrary. However, no hardness result is known for this problem [7,1].

A curious feature of the algorithms proposed in the literature for Parametric Schedulability is that none of them exploit the ordering information in the job-set. In both [5] and [2] polynomial time bounds are derived by observing that the number of relative constraints between any two tasks is bounded, if only strict relative constraints are permitted. In this paper, we explicitly take into account the ordering information to develop two new concepts viz. *Constraint Domination* and *Constraint Orientation*, which in turn are used to develop polynomial time algorithms for testing parametric schedulability in arbitrary network constraints.

Linear programs with at most 2 variables per constraint (LI(2)s) have received quite a bit of attention in the Operations Research community. [15] was the first to observe the correspondence between LI(2)s and graphs; in [6], the Fourier-Motzkin elimination procedure was used to provide a *strongly polynomial* algorithm which to date is the fastest known algorithm for this problem.

5 Algorithms and Complexity

Algorithm (5.1) presents our strategy for testing parametric schedulability.

Function PARAMETRIC-SCHEDULER $(\Upsilon, \mathbf{A}, \vec{\mathbf{b}})$
1: **for** ($i = n$ **down to** 2) **do**
2: ELIM-UNIV-VARIABLE(e_i)
3: **if** (CHECK-INCONSISTENCY()) **then**
4: **return (false)**
5: **end if**
6: PRUNE-CONSTRAINTS()
7: ELIM-EXIST-VARIABLE(s_i)
8: **if** (CHECK-INCONSISTENCY()) **then**
9: **return (false)**
10: **end if**
11: **end for**
12: ELIM-UNIV-VARIABLE (e_1)
13: **if** ($a \leq s_1 \leq b, \quad a, b \geq 0$) **then**
14: `Valid Parametric Schedule Exists`
15: **return**
16: **else**
17: `No Parametric Schedule Exists`
18: **return**
19: **end if**

Algorithm 5.1: A Quantifier Elimination Algorithm for determining Parametric Schedulability

Algorithm (5.2) describes the procedure for eliminating the universally quantified execution variable $e_i \in [l_i, u_i]$. The Fourier-Motzkin elimination technique discussed in [20] represents one implementation of ELIM-EXIST-VARIABLE. In general, any polyhedral projection method suffices. In our work, we assume that the Fourier-Motzkin procedure is used to eliminate the existentially quantified (start-time) variables [1]. When a variable (start-time or execution time) is eliminated, inconsistencies and/or redundancies could result.
CHECK-INCONSISTENCY() identifies inconsistencies and declares the system to be infeasible, while PRUNE-CONSTRAINTS() identifies redundancies and eliminates them.

[1] A detailed exposition of the Fourier-Motzkin elimination procedure is available in [20].

Function ELIM-UNIV-VARIABLE $(\mathbf{A}, \vec{\mathbf{b}})$
1: Substitute $e_i = l_i$ in each constraint that can be written in the form $e_i \geq ()$
2: Substitute $e_i = u_i$ in each constraint that can be written in the form $e_i \leq ()$

Algorithm 5.2: Eliminating Universally Quantified variable $e_i \in [l_i, u_i]$

The correctness of Algorithm (5.2) has been argued in [5], while the correctness of the Fourier-Motzkin procedure is discussed in [13].

We point out that Algorithm (5.1) is similar to the ones outlined in the literature. The difference is the implementation of procedures PRUNE-CONSTRAINTS() and the analysis provided in §5.1.

5.1 Analysis

Observe that the procedure ELIM-UNIV-VARIABLE() does not increase the number of constraints. However, ELIM-EXIST-VARIABLE() has the potential to increase the number of constraints substantially. Assuming that the Fourier-Motzkin elimination algorithm is used, the elimination of k start-time variables, could result in the creation of as many as m^{2^k} constraints. One such pathological example is provided in [13]. In [5] and [18], it was pointed out that "standard" constraints are closed under Fourier-Motzkin elimination i.e. the elimination of an existential variable results in the set of constraints staying standard. Using the notation, in [18], this corresponds to saying that contracting a vertex of the flow graph representing the constraint set, does not destroy the its graph structure. Since a graph has at most $O(n^2)$ edges at all times, the polynomiality of the algorithm for standard constraints follows.

In our case though, there is no obvious way to either represent the set of constraints or bound their number under Fourier-Motzkin elimination, since in addition to relative constraints, we also have sum constraints, as discussed in Section §3. We make the following observation:

Observation 51 *Network constraints are closed under Fourier-Motzkin elimination, using Algorithm (5.1).*

Observation (51) follows from the fact an existential variable i.e. a start time variable is eliminated only *after* the corresponding execution time variable has been eliminated. Consequently, its elimination results in a network constraint between two other jobs. For instance consider the following constraint set:

1. $s_3 \leq s_1 + e_1 + 14$;
2. $s_1 + s_3 \leq 22$;
3. $s_2 + 22 \leq s_3 + e_3$;
4. $e_3 \in [3, 5]$.

The elimination of e_3 results in:

1. $s_3 \leq s_1 + e_1 + 14$;
2. $s_1 + s_3 \leq 22$;
3. $s_2 + 19 \leq s_3$;

The elimination of s_3 (by pairing off constraints in which s_3 occurs with opposite polarity) gives rise to the following set of constraints:

1. $s_2 + 19 \leq s_1 + e_1 + 14$;
2. $s_2 + 19 \leq 22 - s_1$;

The key point is that the network structure is preserved, under the elimination. Observe that if e_3 were not eliminated, prior to eliminating s_3 the closure claim of Observation (51) no longer holds.

Let S_{ij} denote the set of constraints between the two jobs J_i and J_j) $(i < j)$. We now present an informal overview on the nature of constraints $l \in S_{ij}$. Informally, a relative constraint between two jobs either specifies increased separation between them or decreased separation. For instance, the constraint $s_1 + 8 \leq s_2$ specifies increased separation, while the constraint $s_2 \leq s_1 + 17$ specifies decreased separation. An aggregate constraint either pushes the jobs to the left or to the right. For instance the constraint $s_1 + s_2 \leq 7$ pushes jos J_1 and J_2 to the left i.e. towards 0, while the constraint $s_3 + s_4 \geq 8$ pushes jobs J_3 and J_4 towards the right, i.e. towards L.

To proceed with our analysis, we need the following definitions. We associate a type with every constraint, specifying whether it a relative constraint or an aggregate constraint.

Definition 4. *Sum (Aggregate) constraint: A network constraint of the form $a.s_i + b.s_j \leq ()$ is said to be a sum constraint if both a and b have the same sign.*

For instance, $s_1 + s_2 \leq 7$ and $-3.s_1 - 4.s_2 \leq -9$ are sum constraints.

Definition 5. *Difference (Relative) constraint: A network constraint of the form $a.s_i + b.s_j \leq ()$ is said to be a difference constraint, if a and b have opposite signs.*

For instance, the constraint $s_1 - s_2 \leq -4$ is a difference constraint.

Definition 6. *Constraint orientation (Right): A constraint $l \in S_{ij}$ is said to have a* right orientation *if it specifies increased separation between J_i and J_j (in case of difference constraints), or pushes both jobs to the right (in case of sum constraints).*

Definition 7. *Constraint orientation (Left): A constraint $l \in S_{ij}$ is said to have a* left orientation *if it specifies decreased separation between J_i and J_j (in case of difference constraints), or it pushes both jobs to the left (in case of sum constraints).*

For instance, the constraint $s_1 + e_1 + 4 \leq s_2$ specifies that Job J_2 should start at least 4 units *after* J_1 finishes. Since it specifies increased separation, it is has a right orientation. Likewise, the constraint $s_1 + s_3 \leq 12$ requires that J_1 and J_3 move leftward and hence has a left orientation. Using the flow graph terminology in [18], a forward edge in the constraint graph has a right orientation and a backward edge has a left orientation.

Every constraint $l \in S_{ij}, \forall i, j = 1, \ldots, n$ has an orientation, on account of the total ordering on the job-set. Thus a network constraint between job J_1 and J_5 (say) has the effect of either drawing them together or pushing them apart. *This is not true, if there is no ordering on the job-set.* The total ordering on the start time variables implies that all these variables have an interpretation on the same real axis $[0, L]$. In the absence of the total order, each variable has to be interpreted on its own axis. Also see [6].

Definition 8. *Comparable constraints: Two constraints $l_1, l_2 \in S_{ij}$ are said to be comparable if they have the same orientation and type.*

Note that only constraints between the same set of jobs are comparable, i.e. a constraint $l \in S_{13}$ and a constraint $l' \in S_{12}$ are not comparable, regardless of their orientation and type.

Constraint comparability is an *equivalence relation* partitioning the set of constraints between two jobs S_{ij} into the following four categories:

1. Difference Constraint with left orientation (S^1_{ij});
2. Difference Constraint with right orientation (S^2_{ij}).
3. Sum Constraint with left orientation (S^3_{ij});
4. Sum Constraint with right orientation (S^4_{ij});

Definition 9. *Constraint domination: A constraint l_1 is said to dominate another constraint l_2 if and only if they are comparable and $l_1 \Rightarrow l_2$ i.e. l_2 is satisfied, whenever l_1 is satisfied ($l_1, l_2 \in S_{ij}$).*

In some sense, the domination relationship attempts to identify constraints which are redundant. For instance, $s_1 - s_2 \leq -4$ is clearly dominated by $s_1 - s_2 \leq -8$, since if the latter constraint is satisfied, the former is trivially met. The interesting case is the comparison between constraints in which there exist execution time variables. Consider the two comparable constraints:

1. $l_1 : s_1 - s_2 \leq -4$
2. $l_2 : s_1 - s_2 \leq -e_1; e_1 \in [3, 5]$.

Observe that l_2 still dominates l_1; the parametric schedulability query is:
$\exists s_1 \forall e_1 \in [3, 5] \exists s_2 \ldots \mathbf{A}.[\boldsymbol{s}, \boldsymbol{e}] \leq \boldsymbol{b}$?
Since the query is true for *all values of e_1 in the range $[3, 5]$, the constraint $s_1 - s_2 \leq -4$ is subsumed by the constraint $s_1 - s_2 \leq -e_1; e_1 \in [3, 5]$.* This holds true for every pair of comparable constraints l_1 and l_2, i.e. either $l_1 \Rightarrow l_2$ or $l_2 \Rightarrow l_1$. *In other words, the domination relationship imposes a total order on each set of comparable constraints between two jobs. We use lexicographical ordering to break ties.* It follows that in each equivalence class of the partition imposed by the comparability relationship there is a unique constraint that dominates all other constraints in that class [8].

Definition 10. *The unique elements $\kappa_{ij}^k \in S_{ij}^k, k = 1, \ldots, 4$, which dominate all the other constraints in their respective partitions are called the* dominators *of that partition.*

Lemma 1. *If a constraint l_1 dominates another constraint l_2, then eliminating l_2 from the set of constraints, does not alter the parametric schedulability of the system i.e. if the constraint system has a parametric schedule with l_2 then it has a parametric schedule without l_2 and vice versa.*

In other words, we can eliminate l_2 from the constraint set and test for parametric schedulability on the reduced set of constraints. In fact, it suffices to retain the 4 dominators between each pair of jobs, since all other constraints are redundant.

Proof: *We provide a proof for the case in which both l_1 and l_2 belong to the set S_{ij}^1. The other three cases, viz. $l_1, l_2 \in S_{ij}^2$,$l_1, l_2 \in S_{ij}^3$, and $l_1, l_2 \in S_{ij}^4$ can be proved in identical fashion.*

1. *Let the initial system have a parametric schedule - We need to prove that the system will continue to be parametrically schedulable, even after l_2 is eliminated. But this is obvious, since we are reducing the number of constraints! (If A_1 is the feasible region, before the removal of l_2 and A_2 is the feasible region after the removal of l_2, then $A_1 \subseteq A_2$)*
2. *Let the initial system be parametrically unschedulable - We need to show that the removal of l_2 does not make the system schedulable. From our assumption $l_2 \in S_{ij}^1$, i.e. l_2 is a difference constraint that specifies decreased separation between the jobs J_i and J_j. Since l_1 dominates l_2, the separation specified by l_1 is clearly smaller than the separation specified by l_2. Let us assume the contrary and suppose that the constraint system $\mathbf{A'}$ resulting from the elimination of l_2 is parametrically schedulable. This means that there do not exist start times s_i, s_j and execution times e_i which could depend upon s_i and e_j which could depend upon both s_i and s_j, such that a negative cost loop is created. (The system is infeasible if and only if there is such a negative cost loop [3,15,17].) Essentially, we are saying that $\ldots \forall s_i \exists e_i \in [l_i, u_i] \forall s_j \exists e_j \in [l_j, u_j] \ldots \neg(\{\mathbf{A}.[\boldsymbol{s}, \boldsymbol{e}] \leq \boldsymbol{b}\} - \{\mathbf{l_2}\})$ is false. This means that the constraints imposed by the sets S_{ij}^2, S_{ij}^4 can co-exist with l_1.Now consider what happens when l_2 is added to $\mathbf{A'}$. l_2 cannot decrease the separation between jobs J_i and J_2 any more than what is specified by l_1. Consequently if a negative loop could not be created using l_1, then such a loop definitely cannot be created using l_2, i.e. the constraint system stays parametrically schedulable, contradicting the assumption that it was not.*

□

This leads us directly to:

Lemma 2. *There are at most 4 non-redundant constraints between any 2 jobs; hence the total number of non-redundant constraints is at most $O(n^2)$.*

It follows that ELIM-EXIST-VARIABLE() takes at most $O(n^2)$ time, since each start-time variable is part of at most $O(n)$ relationships. PRUNE-CONSTRAINTS() basically performs the equivalent of finding the 4 maxima between each pair of start-times and hence the total time taken is proportional to the number of edges, which is $O(n^2)$. Checking the consistency of the resulting constraints can likewise be carried out in $O(n^2)$ time.

Since all the above functions are called at most $O(n)$ times, the above analysis leads to the following conclusion.

Theorem 1. *Algorithm (5.1) can be implemented to run in $O(n^3)$ worst-case time.*

Proof: *Follows from the discussion above.* □

References

1. Eric Allender. Personal Communication.
2. Seonho Choi. Dynamic dispatching of cyclic real-time tasks with relative time constraints. *JRTS*, pages 1–35, 2000.
3. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill Book Company, 6th edition, 1992.
4. M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinooy Kan, editors. *Deterministic and Stochastic Scheduling.* Reidel, Dordrecht, 1982.
5. R. Gerber, W. Pugh, and M. Saksena. Parametric Dispatching of Hard Real-Time Tasks. *IEEE Transactions on Computers*, 1995.
6. Dorit S. Hochbaum and Joseph (Seffi) Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, December 1994.
7. D.S. Johnson. Personal Communication.
8. Kenneth Kalmanson. *An Introduction to Discrete Mathematics and its Applications.* Addison–Wesley, 1986.
9. S. T. Levi, S. K. Tripathi, S. D. Carson, and A. K. Agrawala. The `Maruti` Hard Real-Time Operating System. *ACM Special Interest Group on Operating Systems*, 23(3):90–106, July 1989.
10. D. Mosse, Ashok K. Agrawala, and Satish K. Tripathi. Maruti a hard real-time operating system. In *Second IEEE Workshop on Experimental Distributed Systems*, pages 29–34. IEEE, 1990.
11. M. Pinedo. *Scheduling: theory, algorithms, and systems.* Prentice-Hall, Englewood Cliffs, 1995.
12. Manas Saksena. *Parametric Scheduling in Hard Real-Time Systems.* PhD thesis, University of Maryland, College Park, June 1994.
13. Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1987.
14. M. Shaked and G. Shanthikumar, editors. *Stochastic Scheduling.* Academic Press, San Diego, 1994.
15. Robert Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, October 1981.
16. John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo, editors. *Deadline Scheduling for Real-Time Systems.* Kluwer Academic Publishers, 1998.

17. K. Subramani. *Duality in the Parametric Polytope and its Applications to a Scheduling Problem*. PhD thesis, University of Maryland, College Park, July 2000.
18. K. Subramani and A. K. Agrawala. A dual interpretation of standard constraints in parametric scheduling. In *The Sixth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, September 2000.
19. K. Subramani and A. K. Agrawala. The static polytope and its applications to a scheduling problem. 3^{rd} *IEEE Workshop on Factory Communications*, September 2000.
20. V.Chandru and M.R. Rao. Linear programming. In *Algorithms and Theory of Computation Handbook, CRC Press, 1999*. CRC Press, 1999.

A Logical Framework for Knowledge Sharing in Multi-agent Systems

Kaile Su[1,2], Xudong Luo[1], Huaiqing Wang[2],
Chengqi Zhang[3], Shichao Zhang[1], and Qingfeng Chen[1,3]

[1] Inst. of Logic and Cognition, Zhongshan Univ., Guangzhou, 510675, P. R. China
[2] Dept. of Information System, City Univ. of Hong Kong, Hong Kong
[3] School of Comput. Sci. and Math., Deakin Univ., Geelong, VIC 3217, Australia

Abstract. The issue of knowledge sharing has been an important topic in multi-agent research. Knowledge sharing leads to that agents analyze, judge and synthesize the told information so as to make agents' own knowledge. To match these applications, this paper builds a logical framework for knowledge sharing among agents. We develop a multi-modal logic for reasoning about both agents' knowledge and told information. For formalizing the relationship between knowledge and told information, we present a framework of semantics, with respect to which a sound and complete proof theory is given.
Keywords: logic, autonomous agents, philosophical foundations

1 Introduction

Currently, the research area of multi-agent systems is an active branch in artificial intelligence. In 1988, Bond and Gasser [1] defined a multi-agent system as a loosely coupled network of autonomous entities called agents which have individual capabilities, knowledge and resources, and which interact to share their knowledge and resources, and to solve problems being beyond their individual capabilities. From this definition, we can see that the issue of knowledge sharing is an important topic in multi-agent research. Therefore, many contributions have been made to this topic in the area of multi-agent systems, such as the knowledge sharing effort [14] and Knowledge Interchange Format (KIF) [15].

However, the issue of knowledge sharing among agents that usually have peculiarities of their knowledge and reasoning properties in the viewpoints of one agent has not been handled yet. We can use the following example to make clear what we exactly mean here. An agent i may believe that another agent j's knowledge or belief is *veridical*, *i.e.*, in terms of standard multi-modal language, $K_i(K_j\alpha \Rightarrow \alpha)$, for all formulas α. Only in this case, agent i is willing to inherit all knowledge of agent j and is ready to accept told information from agent j. So, it is important and interesting to present a reasoning model which enables agents to explicitly express, in their knowledge, both what they have been told and some peculiarities of their colleagues' knowledge and reasoning properties. In this paper, we will tackle this issue.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 561–570, 2001.
© Springer-Verlag Berlin Heidelberg 2001

On the other hand, although there has been a flourishing of epistemic logics formalizing multi-agent knowledge, belief, desire and intention, by using various kinds of possible-worlds semantics, despite their varieties of semantics, these logics usually assume a fixed epistemic logic for a certain class of modalities, and therefore all agents have the same reasoning properties. This is perhaps because the philosophical community, where various epistemic logics were firstly proposed, attacked and defended, has tended to concentrate on the single agent case. However, the case our formalism handles is about multiple heterogeneous agents.

In addition, the problem tackled in this paper is also interesting even in the case of fixing an epistemic logic or a class of epistemic properties. There is a few modal logics combining the notions of an agent's belief (or knowledge), desire and intention; and there even exist general methods of fusing different modal logics (see [4]). However, so far there has not been yet one combining the notions of an agent's knowledge and told information, which is thought surprisingly subtle whenever more than one agent is in the picture [5].

As a first step, we focus on the following epistemic properties:

- *Veridicality.* Only true things are known.
- *Introspection.* An agent is aware of what it knows and of what it does not know.
- *Consistency.* An agent's knowledge is non-contradictory.

Note that, by considering different combinations of the above properties, the well-known modal logics **S5, K, T, K45** and **KD45**, have been constructed and widely discussed.

In this paper, we aim to develop a model for reasoning about not only multi-agent knowledge and told information but also some key reasoning properties of agent's knowledge, including the veridicality and the introspection. We allow the explicit mention of the introspection and the veridicality of an agent's reasoning about its own knowledge, so that our language has formulas indicating which logic of **K**, **T**, **S5**, **K45** is used by an agent.

The rest of this paper is organized as follows. In next section, we review some basic concepts and notations of modal logic, which we will use through this paper. In Section 3, based on the analysis of relationship between knowledge and told information, we define the framework of semantics for our logic. Section 4 addresses the issue of proof theory of our logic. Finally, we discusses the importance of our logic framework in Section 5 and summarizes our contributions in Section 6.

2 Preliminaries

In this section, we recall some basic concepts and notations related to modal logics of multi-agents' knowledge, which we shall use throughout the paper.

The language of those logics is propositional logic augmented by the modal operators $K_1, \cdots, K_n$, where $K_i\phi$ can be read *"agent i knows (or believes) ϕ".*

We denote this language by $\mathcal{L}_n$. For convenience, we define *true* as an abbreviation for a fixed valid propositional formula, say $p \vee \neg p$, where p is primitive proposition. We abbreviate $\neg true$ by $false$.

According to [5], semantics of these formulas can be given by means of *Kripke structure* [8], which formalize the intuition behind possible worlds. A Kripke structure is a tuple $(W, \pi, \mathcal{K}_1, \cdots, \mathcal{K}_n)$, where W is a set of *worlds*, π associates with each world a truth assignment to the primitive propositions, so that $\pi(w)(p) \in \{\mathbf{true}, \mathbf{false}\}$ for each world w and primitive proposition p, and $\mathcal{K}_1, \cdots, \mathcal{K}_n$ are binary accessibility relations. By convention, $\mathcal{K}_i^M$ and π^M are used to refer to the $\mathcal{K}_i$ relation and the π function in the Kripke structure M, respectively. We omit the superscript M if it is clear from context. Finally, we define

$$\mathcal{K}_i(w) = \{w' \mid (w, w') \in \mathcal{K}_i\}.$$

That is, $\mathcal{K}_i(w)$ is the set of worlds agent i considers possible in world w.

A *situation* is a pair (M, w) consisting of a Kripke structure and a world w in M. By using situations, we can inductively give semantics to formulas as follows: for primitive propositions p,

$$(M, w) \models p \text{ iff } \pi^M(w)(p) = \mathbf{true}.$$

Conjunctions and negations are dealt with in the standard way. Finally,

$$(M, w) \models \mathcal{K}_i\alpha \text{ iff for all } w' \in \mathcal{K}_i^M(w),\ (M, w') \models \alpha.$$

Thus, an agent i knows α if α is true in all situations that the agent considers possible.

Note that the Kripke structure M is fixed in the above inductive interpretation. However, it is interesting that our interpretation, as given in the next section, is not in the case.

3 Non-standard Interpretation

This section presents the framework of semantics for our logic. We focus two classes of formulas, *i.e.*, those of the form $K_i\phi$, and those of the form $T_i\phi$, which is read "*agent i is told* ϕ". We use Kripke structures $(W, \pi, \mathcal{K}_1, \cdots, \mathcal{K}_n)$ and the formulas of the form $T_i\phi$ are interpreted by using the accessibility relation $\mathcal{K}_i$ as usual, whereas the interpretation of the formulas of the form $K_i\phi$ is given by the analysis of relationship between knowledge and told information.

3.1 Language

The language we use, denoted by $\mathcal{L}_n^T(\mathbf{VI})$, is $\mathcal{L}_n$ augmented by one class of modalities T_i and two classes of special proposition constants: $\mathbf{I}_i$, and $\mathbf{V}_i$ $(1 \leq i \leq n)$, corresponding to the reasoning properties: the introspection and the veridicality, respectively. Thus, the formula $\mathbf{I}_i$ says that agent i is able to respect his knowledge; the formula $\mathbf{V}_i$ indicates that the agent knows only true things.

We note that it is unnecessary for the language to include some other constants to express the consistency of agent i's knowledge since we can use the formula $\neg K_i false$.

3.2 For Formulas of the Form $T_i\phi$

Now we indicate how to interpret formulas of $\mathcal{L}_n(\mathbf{VI})$. Let $\models_N$ be the satisfaction relation we are going to define. We here use standard Kripke structures $(W, \pi, \mathcal{K}_1, \cdots, \mathcal{K}_n)$ and situations. The key point is that the accessibility relation $\mathcal{K}_i$ in Kripke structure is, however, no longer related to agent i's knowledge; in a each situation (M, w), the syntactic counterpart of those situations (M, w'), where $w' \in \mathcal{K}_i(w)$, is what agent i has been told rather than his knowledge.

Thus, given a situation (M, w), we define

$(M, w) \models_N T_i\phi$ iff for all $w' \in \mathcal{K}_i(w)(M, w') \models_N \phi$.

According to the above definition, each modality T_i need not satisfy veridicality axioms and introspection axioms. Intuitively, the reason for non-introspection is clear; we can certainly tell or not tell you something without telling you that we have told or not told it. Assuming that the external state of agent i's remains unchanged since the agent was told, it seems that the veridicality depends only on the honest of the teller. However, we argue that even if the teller is honest, the veridicality of modality T_i is not necessarily true for i-subjective sentences since the subjective state of agent i described by i-subjective sentences can be changed after agent i is told something. For example, once agent i is told "p is true but you do not know it", this sentence does not hold any longer, because agent i's being told changes agent i's subjective state.

3.3 For Formulas of the Form $K_i\phi$

The relationship between agents' knowledge and the information that agents have been told has received a great deal of attention [11], and is thought of surprisingly subtle whenever more than one agent is considered in the picture [5]. Before interpreting the formulas of the form $K_i\phi$, we give some informal remarks on the relationship. Generally, the told information may be subject to noise; and thus, in order to obtain their own knowledge, agents need to refine the told information via various kinds of refining processes. Of these refining processes, we focus on the following three cases for the purpose of this paper:

- If agent i is veridical, then the agent has the refining process of deleting the false pieces of the told information. Thus, even if agent i is told some fact p, agent i does not necessarily know p whenever p is false and agent i is veridical.
- There are, for each agent, some special pieces of information, usually concerning the agent himself, that are straightforward accessible just by his/her feeling or observation rather than by his/her or other agents' reasoning. Agents believe or refute such pieces of information firmly, no matter what has been told to them. For example, if you feel cold, then you know it, and

you still know it after you are told that you feel warm. The introspection property here is thought of as such information; if agent i is able to introspect, then agent i must know it, no matter what has been told to him.

- If agent i is introspective, then the agent has the refining process of introspection. This process is surprisingly dependent on the veridicality of agent i, which leads the relationship between the told information and the agent's knowledge somewhat subtle. In the absence of the veridicality, the *introspection* of agent i means that if agent i is (resp. is not) told something, then the agent must know he/she is (resp. is not) told it. In the presence of veridicality, agent i pays his/her attention only to true things. The *introspection* of agent i thus indicates that (a) if agent i is told a proposition p which is true, then agent i knows he/she is told the fact, and (b) if a proposition p is false, then agent i knows that he/she is not told p even though he is indeed told p. Nevertheless, in both the cases, we have that if agent i knows something, then he/she knows he/she knows it, and if he/she does not know it, then he/she knows he/she does not. Therefore, the meaning of *introspection* we use here is consistent with that in the literature.

Now let us consider the interpretation of the formulas of form $K_i\alpha$. It is crucial to ask what situations agent i should think possible from the standpoint of what he/she actually knows after checking and cogitating on the told information. Let $\mathcal{S}_i(M, w)$ be the set of all such possible situations for the given situation (M, w). Then, we interpret formulas of the form $K_i\alpha$ in this way:

$$(M, w) \models_N K_i\alpha \text{ iff for all } (M', w') \in \mathcal{S}_i(M, w),\ (M', w') \models_N \alpha.$$

We now figure out what $\mathcal{S}_i(M, w)$ should be. There are four cases according to the logical style of agent i in the situation (M, w). Firstly, if the logical style is **K**, *i.e.* the value of $\pi(w)$ is **false** at both $\mathbf{I}_i$ and $\mathbf{V}_i$, then agent i is unable to distinguish his/her knowledge from what has been told, and hence the situations in $\mathcal{S}_i(M, w)$ are exactly those (M, w')'s, where $w' \in \mathcal{K}_i(w)$.

Secondly, assume the logical style is **T**, *i.e.*

$$\pi(w)(\mathbf{I}_i) = \mathbf{false} \text{ and } \pi(w)(\mathbf{V}_i) = \mathbf{true},$$

then, we get $\mathcal{S}_i(M, w)$ by adding the actual world w to each (M, w'), where $w' \in \mathcal{K}_i(w)$. This enables agent i to delete the false pieces of the told information.

Thirdly, suppose the logical style is **K45**, *i.e.*

$$\pi(w)(\mathbf{I}_i) = \mathbf{true} \text{ and } \pi(w)(\mathbf{V}_i) = \mathbf{false}.$$

Assuming agent i thinks situations (M', w') possible from his own knowledge, we have, by the introspection of agent i, that agent i's told information in each of those situations (M', w') is exactly that in the actual situation (M, w), which is semantically represented as

$$\mathcal{K}_i^{M'}(w') = \mathcal{K}_i^{M}(w).$$

On the other hand, as we mentioned above, the introspection property $\mathbf{I}_i$ is such information that if it holds, then agent i must know it, no matter what agent i has been told. Thus, in each of those situations (M', w'), $\mathbf{I}_i$ must hold, hence

$$\pi^{M'}(w')(\mathbf{I}_i) = \mathbf{true}.$$

Based on the above discussions, we define $\mathcal{S}_i(M, w)$ as the set of those situations (M', w'), where $w' \in \mathcal{K}_i^M(w)$ and M' coincides with M but

$$\mathcal{K}_i^{M'}(w') = \mathcal{K}_i^M(w) \text{ and } \pi^{M'}(w')(\mathbf{I}_i) = \mathbf{true}.$$

For convenience, we denote the above M' Kripke structures by

$$M\left[\frac{\mathcal{K}_i(w')}{\mathcal{K}_i(w)}; \frac{\pi(w')(\mathbf{I}_i)}{\mathbf{true}}\right].$$

Finally, let the logical style be **S5**. Then, by considering the veridicality, we should put the actual world w into the set $\mathcal{K}_i^M(w)$, and think that $\mathcal{K}_i^M(w) \cup \{w\}$ were the set of worlds possible from the standpoint of agent i's told information. By considering the introspection, we define $\mathcal{S}_i(M, w)$ in the same way as in the case of **K45**, nevertheless $\mathcal{K}_i^M(w)$, the set of worlds possible from the standpoint of agent i's told information, is replaced by $\mathcal{K}_i^M(w) \cup \{w\}$. In other words, we define $\mathcal{S}_i(M, w)$ as the set of those situations (M', w'), where $w' \in \mathcal{K}_i^M(w) \cup \{w\}$, and M' coincides with M but

$$\mathcal{K}_i^{M'}(w') = \mathcal{K}_i^M(w) \cup \{w\} \text{ and } \pi^{M'}(w')(\mathbf{I}_i) = \mathbf{true}.$$

For convenience, we denote the M' Kripke structures by

$$M\left[\frac{\mathcal{K}_i(w')}{\mathcal{K}_i(w) \cup \{w\}}; \frac{\pi(w')(\mathbf{I}_i)}{\mathbf{true}}\right].$$

To summarize,

$$\mathcal{S}_i(M, w) = \begin{cases} \{(M\ , w') \mid w' \in \mathcal{K}_i^M(w)\}, \\ \qquad \text{if } \pi(w)(\mathbf{I}_i) = \mathbf{false} \text{ and } \pi(w)(\mathbf{V}_i) = \mathbf{false}; \\ \{(M\ , w') \mid w' \in \mathcal{K}_i^M(w)\} \cup \{(M, w)\}, \\ \qquad \text{if } \pi(w)(\mathbf{I}_i) = \mathbf{false} \text{ and } \pi(w)(\mathbf{V}_i) = \mathbf{true}; \\ \{(M\ [\frac{\mathcal{K}_i(w')}{\mathcal{K}_i(w)}; \frac{\pi(w')(\mathbf{I}_i)}{\mathbf{true}}], w') \mid w' \in \mathcal{K}_i^M(w)\}, \\ \qquad \text{if } \pi(w)(\mathbf{I}_i) = \mathbf{true} \text{ and } \pi(w)(\mathbf{V}_i) = \mathbf{false}; \\ \{(M\ [\frac{\mathcal{K}_i(w')}{\mathcal{K}_i(w) \cup \{w\}}; \frac{\pi(w')(\mathbf{I}_i)}{\mathbf{true}}], w') \mid w' \in \mathcal{K}_i^M(w) \cup \{w\}\}, \\ \qquad \text{if } \pi(w)(\mathbf{I}_i) = \mathbf{true} \text{ and } \pi(w)(\mathbf{V}_i) = \mathbf{true} \end{cases}$$

3.4 Satisfaction Relation

Now we formally present our semantics framework by defining inductively the satisfaction relation $\models_N$ between a situation and a formula as follows. Firstly,

$(M,w) \models_N p$ iff $\pi(w)(p) = \mathbf{true}$, for primitive propositions p. Secondly, $(M,w) \models_N \neg\alpha$ iff not $(M,w) \models_N \alpha$, and $(M,w) \models_N \alpha \wedge \beta$ iff $(M,w) \models_N \alpha$ and $(M,w) \models_N \beta$. Finally, $(M,w) \models_N K_i\alpha$ iff $(M',w') \models_N \alpha$ for all $(M',w') \in \mathcal{S}_i(M,w)$.

We remark that according to this semantics, a veridical agent takes the information he/she was told and adds the current world to the set of possibilities to obtain its knowledge. However, it seems difficult to construct such agents because an agent typically does not know which is the current world and it is impossible for an agent to perform this operation of adding the current world. Nevertheless, this does not imply this semantics is counter-intuitive and suspectable. There might be no veridical agent in the real world, but there are agents that do have the veridicality properties from some other agents' viewpoints. For example, if agent j thinks that agent i is veridical, then, from agent j's viewpoint, agent i does not yet know the current world, but agent j knows (or believes) that the current world is one of agent j's possible worlds. Thus, if the agent j supposes that the current world is w and agent i's set of possible worlds (corresponding to its told information) is W, then agent j thinks that agent i's knowledge is determined by the set W adding the supposed current world w.

4 Proof Theory

This section gives the proof theory of our logic. Let $\mathrm{K}_n(T)$ be the well-known system K_n for the modalities T_i instead of K_i. Clearly, the axioms **P** and **K** and the inference rules in the system $K_n(T)$ are valid with respect to the semantics presented above. In addition, in the absence of K_i, the system $\mathrm{K}_n(T)$ is also complete with respect this semantics. Furthermore, we capture the following important properties of the relationship between told information and knowledge.

Theorem 1 *Let ϕ and φ be arbitrary formulas of $\mathcal{L}_n^T(\mathbf{VI})$, and ψ i-objective.*[1] *Then the following formulas are valid, i.e., satisfied in all situations (M,w):*

1. $(K_i\phi \wedge K_i(\phi \Rightarrow \varphi)) \Rightarrow K_i\varphi$
2. $(\neg\mathbf{I}_i \wedge \neg\mathbf{V}_i) \Rightarrow (K_i\phi \Leftrightarrow T_i\phi)$
3. $(\neg\mathbf{I}_i \wedge \mathbf{V}_i) \Rightarrow (K_i\phi \Leftrightarrow (T_i\phi \wedge \phi))$
4. $\mathbf{I}_i \Rightarrow K_i\mathbf{I}_i$
5. $(\mathbf{I}_i \wedge \neg\mathbf{V}_i) \Rightarrow (K_i\psi \Leftrightarrow T_i\psi)$
6. $(\mathbf{I}_i \wedge \mathbf{V}_i) \Rightarrow (K_i\psi \Leftrightarrow T_i\psi \wedge \psi)$
7. $(\mathbf{I}_i \wedge \neg\mathbf{V}_i) \Rightarrow ((T_i\phi \Rightarrow K_iT_i\phi) \wedge (\neg T_i\phi \Rightarrow K_i\neg T_i\phi))$
8. $(\mathbf{I}_i \wedge \mathbf{V}_i) \Rightarrow (((T_i\phi \wedge \phi) \Rightarrow K_iT_i\phi) \wedge ((\neg T_i\phi \vee \neg\phi) \Rightarrow K_i\neg T_i\phi)))$

This theorem is easy to check by using routine method; we omit its proof here. Let $TK_n^{\mathbf{VI}}$ be the system that results from adding the above valid formulas for each i to the axioms for the system $\mathrm{K}_n(T)$. Thus, Theorem 1 implies that $TK_n^{\mathbf{VI}}$ is sound with respect to our semantics.

[1] We define i-objective formulas as Boolean combinations of primitive propositions (except $\mathbf{I}_i$) and formulas of the form $T_j\phi$, $j \neq i$.

Theorem 2 *For each formula ϕ in $\mathcal{L}_n^T(\mathbf{VI})$, there is a formula ϕ' such that in the system $TK_n^{\mathbf{VI}}$, the two formulas are provably equivalent, but the latter does not have any appearance of K_i's.*

The detail of this proof is omitted for limited space.

For convenience, for an arbitrary formula φ in ϕ in $\mathcal{L}_n^T(\mathbf{VI})$, by $\overline{\varphi}$ we denote a formula which provably equivalent to φ and has no appearance of K_is. By the above result it follows immediately the following theorem.

Theorem 3 (Soundness and completeness) *For the formulas in $\mathcal{L}_n^T(\mathbf{VI})$, the system $TK_n^{\mathbf{VI}}$ is a sound and complete axiomatization with respect the semantics above.*

Proof. The proof of the soundness is easy; axioms $\mathrm{TK}_1 \cdots \mathrm{TK}_8$ are valid by Theorem 1 and other axioms and inference rules are trivially true. As for the completeness, assume that Γ is an arbitrary set of consistent formulas. We must prove that Γ is satisfied by some Kripke structure. Let the formula set $\Gamma' = \{\overline{\varphi} \mid \varphi \in \Gamma\}$. By Theorem 2, Γ is provably equivalent to Γ', and it suffices to show Γ' is satisfied by some Kripke structure. But it is well-known that $\mathrm{K}_n(T)$ is complete with respect to general Kripke structures; we therefore conclude that Γ' is satisfied by a Kripke structure. ■

We end this section by pointing out that

$$(\mathbf{I}_i \wedge \mathbf{V}_i) \Rightarrow (K_i \neg T_i \phi \Leftrightarrow (\neg T_i \phi \vee \neg \phi))$$

is valid, and thus the formula $K_i \neg T_i p$ does not necessarily imply $\neg T_i p$ even under the condition $(\mathbf{I}_i \wedge \mathbf{V}_i)$. It follows that Axiom **T** doesn't hold for our extended language, which makes this system novel in contrast to usual combinations of two classes of modalities (or modal logics).

This also sheds some light on the appropriateness of **S5** as a logic of knowledge. The introspection axiom of **S5** has been attacked at some length in the literature of philosophy and artificial intelligence (see, for example, [10]). Our research indicates that

- for veridical and introspective agents, logic $\mathbf{S5}_n$ is an appropriate logic of knowledge; and
- the introspection axiom reasonably stands even if there are other interactive modalities in the picture.

Nevertheless, the veridicality axiom, which was not attacked before, does not necessarily hold in the presence of other modalities with respect to our semantics.

5 Discussions

This section further shows the significance of our logic in the both philosophical and AI respects.

5.1 General Logic of Knowledge

Consider the language $\mathcal{L}_n(\mathbf{VI})$ that consist of those formulas in $\mathcal{L}_n^T(\mathbf{VI})$ without any appearance of modalities T_i. Our semantics can be used to interpret those formulas. This yields a general logic of knowledge such that different agents may have different reasoning styles.

The approach to the above purpose seems to be straightforward via multiple versions of Kripke structures in which different agents can have different types of accessibility relations. However, in such an approach, the reasoning style of a fixed agent, from viewpoints of different agents, remains unchanged. Actually, it is viewed as common knowledge among all agents concerned. This leads to simple classification of agents provided by three properties: veridicality, introspection and consistency. Such a classification is not appropriate for most general systems of multiple agents. For example, very few agents are of veridicality property in every context.

5.2 Only Knowing

The notion of only knowing (see [12]) plays an important role in knowledge representation, which closely related to formalisms of nonmonotonic reasoning, such as default logic [16] and circumscription [13]. Several notions of multi-agent only knowing have been developed under the assumption that all agents use the same logic [9,6,7]. However, in the context of heterogeneous systems, this assumption is no longer appropriate.

Based on the results in this paper, we will propose a general notion of multi-agent only knowing. In our approach, agent i only knowing something results from the agent only being told something, which is easier to formalize because the logic styles for the modality T_is are fixed **K** despite the multi-formity of logics of agents' knowledge. This is also intuitively reasonable; after all, the less is told, the less is known.

6 Conclusion

In this paper, we have developed a model for reasoning about not only agents' knowledge and told information but also some key reasoning properties of agent's knowledge such as the introspection. These reasoning properties are regarded as styles of how an agent transforms the rough information that has been told, into its own knowledge after checking and cogitating on the former. By using Kripke structures abnormally, we have presented a new framework of semantics, and we give a sound and complete proof theory with respect to the semantics.

Future work includes building more realistic models of knowledge and communication, where more parameters of properties of accessibility relations may be taken into account. It is also interesting to consider temporal factors in those models. Moreover, it is worth to explore the notion of only knowing derived by that of only being told, and compare them with those of previous treatments [12,9,6] on the notion of only knowing.

Acknowledgements

The authors appreciate very much the comments from an anonymous reviewer of this paper. This research was supported by the National Natural Science Foundation of China under Grant Nos. 60073056 and 69733020, by the Guangdong Provincial Natural Science Foundation under Grant No.001174, by the Foundation for University Key Teachers by the Ministry of Education, and by the MOE Project of Key Research Institute of Humanities and Social Science in University.

References

1. Bond, A. and Gasser, L. (eds): Readings in Distributed Artificial Intelligence, Morgan Kaufman Publishers, 1988.
2. Cohen, P.R. and Levesque, H.J.: Communication actions for artificial agents. In *Proc. of the First International Conference on Multi-Agent System*, pp. 65-72, San Francisco, CA, 1995
3. FIPA: Specification part 2 - Agent communication language. The text refers to the specification dated 23 Oct 1997.
4. Gabbay, D.M.: Fibred semantics and the weaving of logics, part 1, modal and intuitionistic logics. Journal of Symbolic Logic, **61** (1996) 1057-1120
5. Halpern, J.Y.: Reasoning about knowledge: a survey. In D.Gabbay et al eds., Handbook of Logic in Artificial Intelligence and Logic Programming, **4**. Oxford University Press. pp.1-34, 1995
6. Halpern, J.Y.: A theory of knowledge and ignorance for many agents. Journal of Logic and Computation. **7** (1997) 79-108
7. Wiebe van der Hoek, Jan Jaspars, and Elias Thijisse. A general approach to multi-agent. In M. Ojeda-Aciego *et al.* (Eds.): *JELIA 2000, Lecture Notes in Computer Science*, Vol. **1919**, pages 254-268, Springer-Verlog, Berlin, 2000.
8. Kripke, S.: A semantical analysis of modal logic. I: Normal modal propositional calculi. Z. Math. Logik Grundl. Math. **9** (1963) 67-96.
9. Lakemeyer, G.: All they know: a study in multi-agent autoepistemic reasoning logic. In Proc. of IJCAI'93, pp.376-381, 1993
10. Lamarre, P. and Shoham, Y.: Knowledge, certainty, belief and conditionalisation. In *Principles of Knowledge Representation and Reasoning: Proc. Fourth International Conference (KR'94)*, pp.415-424, 1994
11. Levesque, J.: Foundations of a functional approach to knowledge representation. Artificial Intelligence, **23** (1984) 155-212
12. Levesque, J.: All I know: a study in autoepistemic logic. Artificial Intelligence, **42** (1990) 263-309
13. McCarthy, J.: Circumscription: A form of nonmonotonic reasoning. Artificial Intelligence, **13** (1980) 27-39
14. Musen, M.A.: Dimensions of Knowledge Sharing and Reuse. Computers and Biomedical Research, **25**(1992) 435-467
15. Patil, R.S., Fikes, R.E. and Patel-Schneider, P.F. et al. The DARPA Knowledge Sharing Effort: Progress Report. Readings in Agents, M. N. Huhns and M. P. Singh (eds.), Morgan Kaufmann Publishers, INc., pp. 243-254, 1998.
16. Reiter, R.: A logic for default reasoning, Artificial Intelligence, **13** (1980) 81-132
17. Wooldridge, M.: Verifying that Agents Implement a Communication Language. In *Proceedings of AAAI-99*, pp. 52-57, 1999.

A Lockout Avoidance Algorithm without Using Time-Stamps for the k-Exclusion Problem

Kumiko Obokata, Michiko Omori, Kazuhiro Motegi, and Yoshihide Igarashi

Department of Computer Science, Gunma University, Kiryu, Japan 376-8515
igarashi@comp.cs.gunma-u.ac.jp

Abstract. We propose a lockout avoidance algorithm for the k-exclusion problem on the asynchronous multi-writer/reader shared memory model. This algorithm is the extension of an accelerated version of the n-process algorithm to the k-exclusion problem. The running time by the algorithm for the trying region of any faultless process is bounded by $(n-k)c + O(n(n-k)^2)l$ if the number of stopping process failures is less than k, where n is the number of processes, l is an upper bound on the time between two successive atomic steps for any faultless process, and c is an upper bound on the time that any user spends in the critical region.

key words: asynchronous processes, concurrent computation, k-exclusion, lockout avoidance, lockout freedom, shared memory

1 Introduction

Mutual exclusion is a problem of managing access to a single indivisible resource that can only support one user at a time. The k-exclusion problem is a natural extension of the mutual exclusion problem. In k-exclusion, some number of processes, specified by the parameter k, are allowed to be concurrently inside the critical region, where corresponding users can use the resource. This extension was first defined and solved by Fischer, et al. on the shared memory model [4]. Afek, et al.[1] gave another solution to the k-exclusion problem. Unlike the solution by Fischer et al. this solution does not use a powerful *read-modified-write* primitive. By the use of a *concurrent time-stamp system* [2,3], it requires only bounded read/write shared memory [1].

The n-process algorithm for the mutual exclusion problem was proposed by Peterson [7], and accelerated by Igarashi and Nishitani [5]. The algorithm proposed in this paper is an extension of these algorithms to the k-exclusion problem on the asynchronous multi-writer/reader shared memory model.

The algorithm is immune to stopping failures of fewer than k processes. All logarithms used in this paper are to the base 2. We impose an upper bound l on the time between successive atomic steps of each faultless process in the trying region and the exit region, and an upper bound c on the time that any user spends in its critical region. Let n be the number of processes in a system. The running time by the algorithm is bounded by $(n-k)c + O(n(n-k)^2)l$. When $k = 1$ (i.e., the mutual exclusion case), the running time $(n-1)c+O(n^3)l$ of the

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 571–575, 2001.
© Springer-Verlag Berlin Heidelberg 2001

algorithm proposed in this paper is better than the running time $O(n^2)c+O(n^4)l$ of the n-process algorithm by Peterson [5,7] and as fast as the running time of the accelerated versions by Igarashi and Nishitani [5].

2 Preliminaries

The asynchronous multi-writer/reader shared memory model consists of processes and shared variables. Interactions between a process and its corresponding user are by input actions from the user to the process and by output actions from the process to the user. All communication among the processes is via the shared memory. The model is called an I/O automaton [6].

A user with access to the resource is modeled as being in a critical region. When a user is not involved in the resource, it is said to be in its remainder region. In order to gain admittance to its critical region, a process executes a trying protocol. The duration from the start of execution of the trying protocol to the entrance of the critical region is called the trying region. After the end of the use of the resource by a user, the corresponding process executes an exit protocol. The duration of the execution of the exit protocol is called the exit region. The k-exclusion problem is to devise protocols for at most k processes to be concurrently in the critical regions.

We assume that the n processes are numbered $1, \cdots, n$. Each process i corresponds to user U_i $(1 \leq i \leq n)$. The inputs to process i from user U_i are try_i which means a request by user U_i for access to the resource, and $exit_i$ which means an announcement of the end of the use of the resource by U_i. The output from process i to user U_i is $crit_i$ which means the grant of the resource to U_i, and rem_i which tells U_i that it can continue with the rest of its work.

The following conditions should be satisfied.

(1) There is no reachable system state in which more than k processes are in the critical regions.
(2) If at least one faultless process is in the trying region and less than k processes are in the critical region, then at some later point some process enters its critical region.
(3) If a faultless process is in the exit region, then at some later point the process enters its remainder region.
(4) If all users always return the resource and if the number of faulty processes of the stopping type is at most $f < k$, then any faultless process wishing to enter the critical region eventually does so.

Conditions (1), (2), (3) and (4) above are called respectively k-exclusion, progress for the trying region, progress for the exit region, and k-lockout avoidance.

3 A Lockout Avoidance Algorithm

procedure (n,k)-*EXCL*
shared variables
  for every $s \in \{1, ..., n-k\}$:
    $turn(s) \in \{1, ..., n\}$, initially arbitrary, writable and readable by all processes;
  for every $i \in \{1, ..., n\}$:
    $flag(i) \in \{0, ..., n-k\}$, initially 0, writable by i and readable by all $j \neq i$;
process i
  input actions {inputs to process i from user U_i}:
    try_i, $exit_i$;
  output actions {outputs from process i to user U_i}:
    $crit_i$, rem_i;
  ** Remainder region **
  try_i:
    for $s = 1$ **to** $n-k$ **do**
      begin
        $flag(i) := s$;
        $turn(s) := i$;
        waitfor $[|\{j|j \neq i : flag(j) \geq s\}| \leq n-s-1]$ or $[turn(s) \neq i]$
      end;
  $crit_i$;
  ** Critical region **
  $exit_i$:
    $flag(i) := 0$;
  rem_i;

For each level s ($1 \leq s \leq n-k$), the statement waitfor$[|\{j|j \neq i : flag(j) \geq s\}| \leq n-s-1]$ or $[turn(s) \neq i]$ in (n,k)-*EXCL* is not an atomic step. It consists of $O(n)$ atomic steps. For each s, the contents of the local variable $count_i$ does not necessarily represent the number of other processes that are located at level s or above. Since the shared memory used in this paper is not the *read-modified-write* model, this uncertainty cannot be avoided.

4 Correctness of (n, k)-*EXCL*

We may assume that any process is initially in the remainder region. Throughout this paper, we assume that the number of stopping process failures is always at most $k-1$. In an execution by (n, k)-$EXCL$, process i is said to be a winner at level s if it has left the waitfor statement in the sth loop of the for statement. Note that if a process is a winner at level s then for any $1 \leq t \leq s$, the process is also a winner at level t. For each i ($1 \leq i \leq n$), when process i has entered its exit region, the qualification for the winner of process i is canceled. We omit the proof of the following lemma due to the space limitations.

Lemma 1. *In any reachable system state of* (n,k)*-EXCL, for any* $1 \leq s \leq n-k$ *there are at most* $n-s$ *winners at level* s*.*

The next theorem follows immediately from Lemma 1.

Theorem 1. *(n,k)-EXCL guarantees k-exclusion even if any number of process failures of the stopping type exist.*

It is obvious that (n,k)-$EXCL$ guarantees the progress for the exit region In order to prove the progress for the trying region, and k-lockout avoidance, it is enough to give a time bound for the trying region.

Theorem 2. *Suppose that the number of stopping failures of processes is always at most* $k-1$*. In* (n,k)*-EXCL, the time from when a faultless process enters its trying region until the process enters its critical region is bounded by* $(n-k)c + O(n(n-k)^2)l$*.*

Proof. For each s, $1 \leq s \leq n-k$, define $F(s)$ to be the maximum time from when a faultless process has entered the competition at level s until it becomes a winner at level s. The worst situation for a faultless process at level $n-k$ to reach the critical region is the following case. Consider the case where there are $k+1$ winners at level $n-k-1$ and all of them are competitors including the process itself at level $n-k$, and then the process becomes a loser among them at level $n-k$.

In the worst case, the time to decide k winners at level $n-k$ is at most $(2n+3)l$. The time duration includes 2 atomic steps for setting $flag$ values and setting $turn(n-k)$ at level $n-k$, the time to test the condition in the waitfor statement at level $n-k$, and local computing time for each faultless process to count the number of $flag(j)$'s with values not less than $n-k$ and for checking the value of $turn(n-k)$. Since the number of stopping failures of processes is at most $k-1$, it is not possible that all the k winners are faulty processes. The time from the end of the competition at level $n-k$ until the time when at least one faultless winner resets its flag value to 0 in its exit region is at most $3l+c$. Then the loser at level $n-k$ becomes a winner at the level within $(2n+1)l$ time after the reset of the winner's flag value. Hence, $F(n-k) \leq (4n+7)l+c$.

The worst situation for a faultless process at level s, $1 \leq s \leq n-k-1$, to become a winner at level s is the case where $n-s+1$ processes have just entered the competition at level s, or some of the processes have stopped due to their stopping failures after they had become winners at level s, whereby the faultless process becomes a loser at level s. In this case, the time required to decide the winners at level s is also at most $(2n+3)l$. The faultless loser at level s will become a winner when a new process joins the competition at level s and then sets $turn(s)$ to be its process name, or when at least one winner at the level eventually resets its flag value to 0 in its exit region. In the worst case, it is sufficient to consider the latter case. In order that the loser at level s becomes a winner at that level, it must wait until at least one winner at level s exits from the critical region. Since the number of faulty processes is at most $k-1$, there exists a faultless winner at that level. The quickest winner at level s can reach the critical region within $(n-k-s)(2n+3)l$ time. Hence,

the loser at level s can move up as a winner at level s to level $s+1$ within $F(s) \leq (n-k-s+1)(2n+3)l+(2n+1)l+3l+c = (n-k-s+2)(2n+3)l+l+c$. Thus the time from when a faultless process has entered its trying region until it enters its critical region is bounded by

$$\sum_{s=1}^{n-k} F(s) \leq \sum_{s=1}^{n-k-1} [(n-k-s+2)(2n+3)l+l+c)] + (4n+7)l+c$$
$$\leq (n-k)c + O(n(n-k)^2)l.$$

□

The next theorem fallows immediately from Theorem 2 and Theorem 3.

Theorem 3. *(n,k)-EXCL solves the k-exclusion problem, and it is k-lockout avoidance.*

5 Concluding Remarks

The total size of the shared memory for (n,k)-$EXCL$ is $(n-k)\lceil \log n \rceil + n\lceil \log(n-k+1) \rceil$ bits. We are interested in a problem of whether some modification of these algorithms can reduce the shared memory size. We are also interested in a problem of whether we can speed up our algorithm by simple modification for the case where both k and $n-k$ are $\Theta(n)$ or for the case where k or $n-k$ is $O(1)$. The k-assignment problem is closely related to the k-exclusion problem. We are tempted to solve the k-assignment problem without using concurrent time-stamps by similar techniques.

References

1. Y.Afek, D.Dolev. E.Gafni, M.Merritt, and N.Shavit, "A bounded first-in, first-enable solution to the l-exclusion problem", ACM Transactions on Programming Languages and Systems, vol.16, pp.939–953, 1994.
2. D.Dolev and N.Shavit, "Bounded concurrent time-stamp systems are constructible", *21st Annual ACM Symposium on the Theory of Computing*, New York, pp.454–465, 1989.
3. C.Dwork and O.Waarts, "Simple and efficient bounded concurrent timestamping or bounded concurrent timestamp systems are comprehensible", *24th Annual ACM Symposium on the Theory of Computing*, Victoria, British Columbia, pp.655–666, 1992.
4. M.J.Fischer, N.A.Lynch, J.E.Burns, and A.Borodin, "Resource allocation with immunity to limited process failure", *20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, pp.234–254, 1979.
5. Y.Igarashi and Y.Nishitani, "Speedup of the n-process mutual exclusion algorithm", Parallel Processing Letters, vol.9, pp.475–485, 1999.
6. N.A.Lynch, "Distributed Algorithms", *Morgan Kaufmann*, San Francisco, California, 1996.
7. G.L.Peterson, "Myths about the mutual exclusion problem", *Information Processing Letters*, vol.12, pp.115–116, 1981.

Prefix-Free Languages and Initial Segments of Computably Enumerable Degrees

Guohua Wu*

School of Mathematical and Computing Sciences
Victoria University of Wellington
P.O.Box 600, Wellington, New Zealand
wu@mcs.vuw.ac.nz

Abstract. We study prefix-free presentations of computably enumerable reals. In [2], Calude et. al. proved that a real α is c.e. if and only if there is an infinite, computably enumerable prefix-free set V such that $\alpha = \sum_{\sigma \in V} 2^{-|\sigma|}$. Following Downey and LaForte [5], we call V a prefix-free presentation of α. Each computably enumerable real has a computable presentation. Say that a c.e. real α is simple if each presentation of α is computable. Downey and LaForte [5] proved that simple reals locate on every jump class. In this paper, we prove that there is a non-computable c.e. degree bounding no noncomputable simple reals. Thus, simple reals are not dense in the structure of computably enumerable degrees.

1 Introduction

This paper investigates the relationship between the c.e. degrees (the computational complexity classes under Turing reducibility of sets of positive integers which can be effectively enumerated) and the presentations of computably enumerable reals. Investigation on effectiveness of reals originated in Turing's paper [11], and has now become one of the central topics of computable analysis and algorithmic information theory.

Say that a real α is *computable* if α is the limit of a computable sequence of rationals $\{r_i : i \in \omega\}$ and there is a computable algorithm M such that for all n, $|\alpha - r_{M(n)}| < 2^{-n}$. All algebraic numbers, π, the Euler number e, and all numbers commonly used in numerical analysis are computale. Note that M here controls the rate of the convergence of $\{r_i : i \in \omega\}$. Effectivity of M is crucial because there are computable sequences of rationals converging (non-effectively) to non-computable reals, see Spector [10]. Based on this, Soare [8] defined *computably enumerable reals* as follows:

Definition 1 We call a real α computably enumerable (c.e., for short), if α is the limit of a computable sequence of rationals $\{r_i : i \in \omega\}$.

* This research is supported by the Marsden Funds of New Zealand. The author would like to thank his thesis supervisor, Prof. Rod Downey, for many helpful suggestions.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 576–585, 2001.
© Springer-Verlag Berlin Heidelberg 2001

For simplicity, we only consider reals between 0 and 1. In [8], Soare proved that α is c.e. iff the lower (left) Dedekind cut of α is computably enumerable. For more equivalent definitions of c.e. reals, we need a technical notion whose use is crucial in the investigations of randomness of reals.

Let $\Sigma = \{0,1\}$ be the binary alphabet, and Σ^* be the set of finite binary strings. For $\sigma \in \Sigma^*$, let $|\sigma|$ be the length of σ. $A \subseteq \Sigma^*$ is prefix-free if for any $\sigma_1, \sigma_2 \in A$, if σ_1 is a prefix of σ_2, then $\sigma_1 = \sigma_2$. Say that C is a *Chaitin machine* (self-delimiting Turing machine) if C is a Turing machine processing binary strings such that its program set (domain) $PROG_C = \{\sigma \in \Sigma^* | C(\sigma)$ halts$\}$ is prefix-free. Clearly, $PROG_C$ is computably enumerable (c.e. for short). Conversely, each prefix-free c.e. set of strings is the domain of some Chaitin machine. The *program-size complexity* of the string $\sigma \in \Sigma^*$ (relatively to C) is defined as $H_C(\sigma) = \min\{|\eta| : \eta \in \Sigma^*, C(\eta) = \sigma\}$, where $\min \emptyset = \infty$.

Theorem 1 (Invariance Theorem) There is a Chaitin machine U (called universal) such that for every Chaitin machine C, $H_U(\sigma) \leq H_C(\sigma) + O(1)$.

Note that $PROG_U$ is c.e., but not computable.

Generally, for any prefix-free languages, A say, $\sum_{\sigma \in A} 2^{-|\sigma|}$ converges to a real in $[0,1]$, i.e., $\sum_{\sigma \in A} 2^{-|\sigma|} \leq 1$. This is called the *Kraft's Inequality*. We use 2^{-A} to denote $\sum_{\sigma \in A} 2^{-|\sigma|}$. Thus, by the property of $PROG_U$ mentioned above, we have that 2^{-PROG_U} is a c.e., but noncomputable, real in the unit interval. As usual, we use Ω_U to denote the real 2^{-PROG_U}. Calude, Hertling, Khoussainov and Wang [2], Downey, Hirschfeldt and Nies [4], Kučera and Slaman [6] provide extensive investigations of the randomness of Ω_U.

The following are equivalent definitions of c.e. reals.

Theorem 2 (**Calude et. al.** [2]) For any α in the unit interval, the following are equivalent:

(1) α is computably enumerable.

(2) The left Dedekind cut of α is computably enumerable.

(3) There is an infinite computably enumerable prefix-free set $V \subset \Sigma^*$ such that $\alpha = 2^{-V}$.

(4) There is a computable prefix-free set $V \subset \Sigma^*$ such that $\alpha = 2^{-V}$.

(5) There is a computable increasing sequence of rationals with limit α.

Based on characterization (3), Downey and LaForte [5] proposed the notion of presentations as follows:

Definition 2 (Downey and LaForte [5]**)** Say that $V \subseteq \Sigma^*$ is a prefix-free presentation of c.e. real α if (1) V is a prefix-free c.e. set; (2) $\alpha = 2^{-V}$.

By (4) of Theorem 2, each c.e. real has a computable presentation. Say that a c.e. real α is simple if all presentations of α are computable. Since simple c.e. reals resemble computable reals, it is tempting to conjecture that simple reals might be related to the known classes of c.e. sets resembling computable sets (like the low sets) in the sense of the jump hierarchy. Downey and LaForte [5] refuted this by showing that there are simple c.e. reals which are high.

Theorem 3 (Downey and LaForte [5]**)** There is a simple real α with $\alpha' \equiv_{\mathrm{T}} \emptyset''$.

From the point of view of computability theory, Downey and LaForte's result suggests to asking whether all c.e. degrees contain some simple reals, or whether the degrees of simple reals are dense in the class of all c.e. degrees. In this paper, we prove that answers to these two questions are both negative.

Theorem 4 There is a noncomputable c.e. set A such that A bounds no noncomputable simple reals. That is, for any noncomputable c.e. real $\alpha \in [0,1]$, if $\alpha \leq_{\mathrm{T}} A$, then α has a noncomputable presentation.

Here, $\alpha \leq_{\mathrm{T}} A$ means that there is a (partial) computable functional Φ such that $\alpha = \Phi(A)$, namely, the real $0.\Phi(A;0)\Phi(A;1)\cdots$ is a binary expansioanry of α. Theorem 4 is proved by induction as follows:

Theorem 5 There is a noncomputable c.e. set A s.t. for any $n \geq 0$, for any noncomputable c.e. real $\alpha \in [1-\frac{1}{2^n}, 1-\frac{1}{2^{n+1}}]$, if $\alpha \leq_{\mathrm{T}} A$, then α has a noncomputable presentation.

The main part of this paper is devoted to the proof of the base step $n = 0$:

Theorem 6 There is a noncomputable c.e. set A s.t. for any noncomputable c.e. real $\alpha \in [0, \frac{1}{2}]$, if $\alpha \leq_{\mathrm{T}} A$, then α has a noncomputable presentation.

The successor step of the proof is: Fix A as in Theorem 5, assume that we have Theorem 5 for n. We now consider the case $n+1$. Fix $\alpha \in [1-\frac{1}{2^{n+1}}, 1-\frac{1}{2^{n+2}}]$, $\alpha \leq_{\mathrm{T}} A$ as a noncomputable c.e. real. Then $\beta = 2\alpha - 1 \in [1-\frac{1}{2^n}, 1-\frac{1}{2^{n+1}}]$ and $\beta \leq_{\mathrm{T}} A$ is also a noncomputable c.e. real. By the induction hypothesis, β has a noncomputable presentation V_β. Define $V_\alpha = \{0\sigma : \sigma \in V_\beta\} \cup \{1\}$. Then V_α is c.e. and noncomputable. Prefix-freeness of V_α comes from that of V_β directly. Also we have

$$2^{-V_\alpha} = \sum_{\eta \in V_\alpha} 2^{-|\eta|} = \sum_{\sigma \in V_\beta} 2^{-|0\sigma|} + \frac{1}{2} = \frac{1}{2} \sum_{\sigma \in V_\beta} 2^{-|\sigma|} + \frac{1}{2} = \frac{1}{2}\beta + \frac{1}{2} = \alpha.$$

Therefore, V_α is a presentation of α. □

2 Requirements

In this section we list the requirements necessary to prove Theorem 6. We will construct a c.e. set A satisfying for all i the requirements:

$\mathcal{P}_i$: $A \neq \overline{W}_i$;
$\mathcal{R}_i$: $[\alpha_i = \Phi_i(A) = 2^{-F_i} \in [0, \frac{1}{2}]$ & α_i is c.e. & F_i is prefix-free$] \Rightarrow$ [α_i is computable $\vee$ $(\exists$ noncomputable c.e. $U_i)(U_i$ presents $\alpha_i)$].

Here we assume that $\{W_i\}_{i\in\omega}$ is a listing of all c.e. sets of natural numbers, and $\{(\alpha_i, \Phi_i, F_i)\}_{i\in\omega}$ is a listing of all triples, (α, Φ, F) such that α is a c.e. real, Φ is a partial computable functional, and F is a c.e. set of strings.

Since $\mathcal{R}_i$ is a complicated requirement, we break it into simpler subrequirements $\mathcal{S}_{i,j}$ as follows:

$\mathcal{S}_{i,j}$: $U_i \neq \overline{V}_j$.

Here $\{V_i\}_{i\in\omega}$ is a listing of all c.e. sets of strings in Σ^*.

3 Basic Modules for Each Requirement

We now describe the atomic strategies (*modules*) for each type of requirement $\mathcal{P}_i, \mathcal{R}_i, \mathcal{S}_{i,j}$.

(3.1) The ρ-module for $\mathcal{P}_i$

A $\mathcal{P}$-strategy, ρ say, trying to figure out a number x such that $A(x) \neq \overline{W}_i(x)$, is the standard Friedberg-Muchnik procedure. ρ runs as follows:

(1) Choose a big number, x say.
(2) Wait for $\Phi_i(x)\downarrow$.
(3) Put x into A, and stop.

ρ has two finitary outcomes: w and s, where w denotes the fact that ρ waits at (2) forever, and s denotes the fact that ρ reaches (3) and stop. In both cases, $A(x) \neq \overline{W}_i(x)$, and $\mathcal{P}_i$ is satisfied.

(3.2) The τ-module for $\mathcal{R}_i$

Let $\alpha_i[s]$ be the approximation of α_i at stage s, F_i is a c.e. set of strings presenting α_i. W.o.l.g., we assume that for all s, $\alpha_i[s] \leq \sum_{\sigma\in F_{i,s}} 2^{-|\sigma|} \leq \frac{1}{2}$. Define the distance between α_i and Φ_i^A as follows:

$$\ell(i,s) = \max\{x < s : (\forall y < x)[\Phi_i^A(y)[s]\downarrow]\},$$
$$m\ell(i,s) = \max\{0, \ell(i,t) : t < s\},$$
$$d(i,s) = |\textstyle\sum_{y<\ell(i,s)} \Phi_i^A(y)[s]\cdot 2^{-y} - \alpha_i[s]|,$$
$$md(i,s) = \min\{0, d(i,t) : t < s\}.$$

Say that stage s is i-expansionary if $s = 0$ or if $\ell(i,s) > \max\{m(i,s), |\sigma| : \sigma$ is assigned to some $\mathcal{S}_{i,j}$-strategy$\}$, $d(i,s) < \min\{md(i,s), 2^{-m\ell(i,s)}, 2^{-|\sigma|} : \sigma$ is assigned to some $\mathcal{S}_{i,j}$-strategy$\}$, and $\alpha_i[s] \geq 2^{-F_{i,t}}$ for all $t < s$.

The τ-module has a (*finitary*) outcome 1, in case that there are only finitely many i-expansionary stages, and an (*infinitary*) outcome 0, in case that there are infinitely many i-expansionary stages.

If τ-module has an outcome 0, then τ is responsible to construct an auxiliary c.e. set of strings U_τ such that U_τ is noncomputable, and that U_τ presents α_τ. U_τ is constructed by τ's substategies. Links from τ to these substrategies will be created and removed alternatively in the whole construction. We will also ensure that $\sum_{\sigma\in U_{\tau,s}} 2^{-|\sigma|} \leq \alpha_{\tau,s}$, and infinitely often, we can update U_τ such that $\sum_{\sigma\in U_{\tau,s}} 2^{-|\sigma|} = \alpha_{\tau,s}$. We can do this by Chaitin-Kraft inequality. Thus, we can effectively expand the prefix-free set $U_{\tau,s}$ to a larger one with the prefix-freeness preserved. Since $\alpha_i \in [0, \frac{1}{2}]$, we can always choose the strings for the updating process as 1σ, and leave the strings of the form $00\cdots01$ for the $\mathcal{S}_{i,j}$-strategies, $j \in \omega$. This arrangement avoids the interaction between the diagonalization and the updating process.

(3.3) The η-module for $\mathcal{S}_{i,j}$

The η-module is the "gap-cogap" argument, which is the key part of the proof. η-module requires various parameters such as $\sigma(\eta), r(\eta)$, whose values at the end of stage s will be denoted by $\sigma(\eta)[s], r(\eta)[s]$. η-module has two objectives. One is to find some string σ witnessing $U_\eta \neq \overline{V}_\eta$, and if this fails, η will ensure in a definite way that the real α_η (namely, $\alpha_{\tau(\eta)}$) is computable. In the first case, η itself is satisfied. In the second case, $\mathcal{R}_i$ is satisfied and no further subrequirements $\mathcal{S}_{i,j'}, j' \neq j$, need to be examined.

The η-module consists of the following steps:

(1) Choose a string $\sigma = 00\cdots01$, with $|\sigma|$ big.

(2) Wait for a stage s such that $\sigma \in V_j[s]$. At stage $s+1$ *open a gap* by setting $r[s+1] = 0$, and go to step (3). **Stop updating U_i until this gap is closed.**

(3) Wait for the least i-expansionary stage $t \geq s+1$ such that $\ell(i,t) > |\sigma|$. At stage $t+1$ *close the gap* by performing step (3a) or (3b) according to which case applies:

(3a) (**Successful close**) Suppose that $\alpha_i[t] - \alpha_i[s] > 2^{-|\sigma|}$. Enumerate σ into U_i, update U_i and stop.

(3b) (**Unsuccessful close**) Otherwise, define $r[t+1] = t$ (to preserve $\Phi_i^A[t] \upharpoonright |\sigma|$). Cancel σ, update U_i and go to step (1).

Note that if $\alpha_i = \Phi_i^A$ and F_i is a presentation of α_i, then every gap must be eventually closed, and so U_i will be updated correctly. η-module has three possible outcomes:

w. In this outcome, the η-module opens at most finitely many gaps and never closes one successfully.

Let s be the last stage at which η closes its gap, and σ be chosen at stage $t > s$. Then σ never goes into V_j, because otherwise, η will create another link, which is contrary to the choice of s. Thus, $\overline{V}_j(\sigma) = 1 - V_j(\sigma) = 1 \neq 0 = U_i(\sigma)$. $\mathcal{S}_{i,j}$ is satisfied.

s. There is some stage $s+1$ at which η closes a gap successfully.

In this case, $\sigma = \sigma[s]$ is enuemrated into U_i at stage $s+1$. Suppose that at stage $v_1 < s$, σ was defined as $\sigma[s]$, and at stage $v_2 \in [v_1, s)$, η opened a gap trying to enumerate σ into U_i. By the module, $\alpha_i[s] - \alpha_i[v_2] > 2^{-|\sigma|}$, since the updating process does not perform during the gap, we are eligible to put σ into U_i. Thus, $U_i(\sigma) = 1 \neq 0 = 1 - \overline{V}_j(\sigma)$, $\mathcal{S}_{i,j}$ is satisfied.

g. In this outcome, η opens infinitely many gaps and closes each one unsuccessfully.

Obviously, the limit of the length of the candidates (strings) is ∞. Suppose that the infinitely many gaps are begun at stages $\{t_n\}_{n\in\omega}$ and closed at stages $\{v_n\}_{n\in\omega}$, where $t_0 < v_0 < t_1 < v_1 < \cdots$. The intervals of stages $\{s : t_n \leq t < v_n\}$ are called gaps, since $r = 0$ in these intervals and we are free to enumerate any number into A to make A noncomputable, while the intervals $\{s : v_n \leq s < t_{n+1}\}$ are called cogaps and we have $r = v_n$.

α_i is computable because:

Given n. Choose a stage t large enough such that a gap is begun at stage $t+1$, and the corresponding string σ has length $|\sigma| > n+3$. Since the gap is closed unsuccessfully at stage $v+1 > t+1$, we have $\alpha_i[v] - \alpha_i[t] \leq 2^{-|\sigma|}$. If the next gap is begun at stage $t'+1 > v+1$, then the A-restraint r ensures that $\Phi^A \upharpoonright |\sigma|[v] = \Phi^A \upharpoonright |\sigma|[t']$. Thus,

$$\begin{aligned}
& \alpha_i[t'] - \alpha_i[v] \\
\leq\ & |\alpha_i[t'] - \textstyle\sum_{y<\ell(i,v)} \Phi_i^A(y)[v] \cdot 2^{-y}| + |\sum_{y<\ell(i,v)} \Phi_i^A(y)[v] \cdot 2^{-y} - \alpha_i[v]| \\
\leq\ & |\alpha_i[t'] - \textstyle\sum_{y<\ell(i,t')} \Phi_i^A(y)[t'] \cdot 2^{-y}| + |\sum_{y<\ell(i,t')} \Phi_i^A(y)[t'] \cdot 2^{-y} \\
& - \textstyle\sum_{y<\ell(i,v)} \Phi_i^A(y)[v] \cdot 2^{-y}| + |\sum_{y<\ell(i,v)} \Phi_i^A(y)[v] \cdot 2^{-y} - \alpha_i[v]| \\
\leq\ & 2^{-|\sigma|} + 2^{-|\sigma|} + 2^{-|\sigma|} \\
=\ & 3 \cdot 2^{-|\sigma|}.
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\alpha_i[t'] - \alpha_i[t] &= (\alpha_i[t'] - \alpha_i[v]) + (\alpha_i[v] - \alpha_i[t]) \\
&\leq 3 \cdot 2^{-|\sigma|} + 2^{-|\sigma|} = 2^{-(|\sigma|-2)}.
\end{aligned}$$

Now the argument continues with σ replaced by a new one, σ' say. This establishes the following:

$$\alpha_i - \alpha_i[t] \leq 2^{-(|\sigma|-2)} + 2^{-(|\sigma'|-2)} + \cdots \leq 2^{-(|\sigma|-3)}.$$

Since $|\sigma| > n+3$, we have $\alpha_i - \alpha_i[t] < 2^{-n}$. Thus, α_i is computable.

In this case, $\liminf_s r[s] = 0$, because r is defined as 0 during gaps.

g is an infinitary outcome, while w, s are both finitary.

4 The Priority Tree

Let $\Lambda = \{0, 1, s, g, w\}$ with the order $<_\Lambda$ in the order (from left to right) as listed. T is defined as follows as a subset of the finite sequences of Λ.

(4.1) (The priority tree) (1) Define $T = \{\xi \in \Lambda^{<\omega} : (\forall n)[\xi(3n) \in \{0,1\}, \xi(3n+1) \in \{s,w\}\ \&\ \xi(3n+2) \in \{s,g,w\}]\}$.

(2) Define $[T] = \{h \in \Lambda^\omega : (\forall n)[h \upharpoonright n \in T]\}$, the set of all infinite path through T.

Every node $\tau \in T$ with $|\tau| \equiv 0 \pmod 3$, will be assigned to the $\mathcal{R}$-requirement. The outcome 1 for τ denotes that there are at most finitely many τ-expansionary stages, so τ believes that $\mathcal{R}$ is satisfied. The outcome 0 denotes that there are infinitely many τ-expansionary stages, and that further actions to satisfy $\mathcal{R}$ must be taken by nodes $\eta \supset \tau$.

Every node $\rho \in T$ with $|\rho| \equiv 1 \pmod 3$, will be assigned to the $\mathcal{P}$-requirement.

Every node $\eta \in T$ with $|\eta| \equiv 2 \pmod 3$, will be assigned to some subrequirement $\mathcal{S}_{i,j}$. The η-strategy will be a variant of the basic module described in the section above, having outcomes from $\{s, g, w\}$.

(4.2) Define (partial) functions $i, j : T \to \omega$ and lists L_0, L_1, L_2, by induction on $n = |\xi|$ as follows.

$n = 0$. Define as above, $i(\lambda) = 0$, and set $L_0(\lambda) = L_1(\lambda) = L_2(\lambda) = \omega$.

$n > 0$. Let ξ^- be the immediate predecessor of ξ.

Assume that $i(\xi^-), L_0(\xi^-), L_1(\xi^-), L_2(\xi^-)$ are defined, and that $j(\xi^-)$ is defined if ξ^- is an $\mathcal{S}$-strategy. First define $L_0(\xi), L_1(\xi)$ and $L_2(\xi)$ as those in the next paragraph. If ξ is an $\mathcal{R}$-strategy, define $i(\xi) = \mu i[i \in L_0(\xi)]$, and let $j(\xi)$ be undefined. If ξ is a $\mathcal{P}$-strategy, define $i(\xi) = \mu i[i \in L_1(\xi)]$, and also let $j(\xi)$ be undefined. If ξ is an $\mathcal{S}$-strategy, define $i(\xi) = i_0, j(\xi) = j_0$ where $\langle i_0, j_0\rangle = \mu n[n \in L_2(\xi)]$.

Case 1: $|\xi^-| \equiv 0 \pmod 3$. Adopt the subcase below.

subcase (i): $a = 0$. Define $L_0(\xi) = L_0(\xi^-) - \{i(\xi^-)\}$, $L_1(\xi) = L_1(\xi^-)$, and $L_2(\xi) = L_2(\xi^-)$.

subcase (ii): $a = 1$. $L_0(\xi) = (L_0(\xi^-) - \{i(\xi^-)\}) \cup \{i' : i' > i(\xi^-)\}$, $L_1(\xi) = L_1(\xi^-) \cup \{i' : i' > i(\xi^-)\}$, and $L_2(\xi) = (L_2(\xi^-) - \{\langle i(\xi^-), k\rangle : k \in \omega\}) \cup \{\langle i', k\rangle : i' > i(\xi^-)\ \&\ k \in \omega\}$.

Case 2: $|\xi| \equiv 1 \pmod 3$. Define $L_0(\xi) = L_0(\xi^-)$, $L_1(\xi) = L_1(\xi^-) - \{i(\xi^-)\}$, and $L_2(\xi) = L_2(\xi^-)$.

Case 3: $|\xi| \equiv 2 \pmod 3$. Adopt the subcase below.

subcase (i): $a = s$ or $a = w$. Define $L_0(\xi) = L_0(\xi^-)$, $L_1(\xi) = L_1(\xi^-)$, and $L_2(\xi) = L_2(\xi^-) - \{\langle i(\xi^-), j(\xi^-)\rangle\}$.

subcase (ii): $a = g$. Define $L_0(\xi), L_1(\xi)$ and $L_2(\xi)$ as in *Case 1, subcase (ii).*

This completes the priority assignment.

5 Construction

At the end of the construction, we define the *true path* $f \in [T]$ of the construction. f is approximated via $\delta[s] \in T$ at each stage s, and we will have $f = \liminf_s \delta[s]$. During stage s, we define $\delta[s]$ via a series of substages $t < s$ at which we define a string δ_t such that $\delta_t \subseteq \delta_{t+1}$ and $\delta[s]$ is the last δ_t defined. Only string δ_t is allowed to act at substage t.

(5.1) (Genuine stages) For $\xi \in T$,

(1) Say that $s + 1$ is a ξ-stage if $\xi \subseteq \delta[s + 1]$. In addition 0 is a ξ-stage. Let S^ξ be the set of all ξ-stages.

(2) Say that $s + 1$ is a genuine ξ-stage if $\xi = \delta_t$ for some substage t of stage $s + 1$, where δ_t will be defined during the construction. Let G^ξ be the set of genuine ξ-stages.

(5.2) (Expansionary stages) Fix τ. At stage $s+1$, define the length functions as:

$\ell(\tau, s) = \max\{x < s : (\forall y < x)[\Phi^A_\tau(y)[s] \downarrow]\}$,
$m\ell(\tau, s) = \max\{0, \ell(\tau, t) : t \le s\ \&\ t + 1 \in S^\tau\}$.

Also define the distance function as:

$d(\tau, s) = |\sum_{y<\ell(\tau,s)} \Phi^A_\tau(y)[s] \cdot 2^{-y} - \alpha_\tau[s]|$,
$md(\tau, s) = \min\{0, d(\tau, t) : t \le s\ \&\ t + 1 \in S^\tau\}$.

Say that stage v is τ-expansionary stage, if $v = 0$, or if $v = s + 1 \in S^\tau$, $\ell(\tau, s) > \max\{m\ell(\tau, s), |\sigma| : \sigma$ is ever assigned to some $\mathcal{S}_{i(\tau),j}$-strategy$\}$, $d(\tau, s) < \min\{md(\tau, s), 2^{-m\ell(\tau,s)}, 2^{-|\sigma|} : \sigma$ is ever assigned to some $\mathcal{S}_{i(\tau),j}$-strategy$\}$, and $\alpha_\tau[s] > 2^{-F_{\tau,t}}$ for all $t < s, t \in S^\tau$.

Suppose that at stage s, an $\mathcal{S}$-strategy η opens a gap, then we shall construct a link (τ, η) with bottom η and top $\tau = \tau(\eta)$. By the construction of T, we have $\tau^\frown\langle 0\rangle \subseteq \eta$ and $F(\eta)[s] = g$. The link (τ, η) remains in force until either η closes the gap (either a successful or a unsuccessful close), or η is initialized. If at stage $s + 1$, there is a link (τ, η) and there is a substage t of stage $s + 1$ such that $\delta_t = \tau$ and $\delta_{t+1} = \eta$, then we say that the link (τ, η) is travelled at stage $s + 1$. The purpose of the link is to allow the flow of control of the construction to pass directly from the top node τ to the bottom node η at τ-expansionary stages, to close the gap. The hypothesis $\alpha_\tau = \Phi_\tau(A)$ guarantees that each link will be travelled (at most once) and the corresponding gap will be closed.

Note that travelling the link (τ, η) may lead the construction to pass over some nodes $\xi, \tau \subset \xi \subset \eta$ of lower global priority, i.e., $i(\xi) > i(\eta) = i(\tau)$, which would otherwise receive attention before η if we simply descended the tree without using links. For such ξ, $s+1$ is a ξ-stage, but not a genuine ξ-stage. This can result in the failure of ξ. Thus, to consider whether a given ξ-strategy succeeds or not, we must examine the genuine ξ-stages rather than merely the ξ-stages.

(5.3) (Construction)

The construction is as follows:

Stage $s = 0$. Initialize all nodes $\xi \in T$. Define $\delta[0] = \lambda$.

Stage $s+1$. The construction will proceed by substages $t \leq s+1$. We refer to substage t of stage $s+1$ as stage $(s+1, t)$. The value of parameter p at the end of substage t will be denoted by p_t, rather that the complete specification $p_t[s+1]$. At the end of substage t we shall have $\delta_t \in T$ such that $\delta_t \subseteq \delta_{t+1} \subseteq \delta[s + 1]$, for $t \leq s$, and only δ_t may act during substage t. At the end of stage $s + 1$, if $s + 1$ is a τ-expansionary stage, and there is no link with top τ created at stage $s + 1$, where τ is an $\mathcal{R}$-strategy, then perform the updating process for τ by enumerating into U_τ approriate strings to make $\sum_{\sigma\in U_{\tau,s+1}} 2^{-|\sigma|} = \alpha_{\tau,s+1}$. This process keeps the resulted c.e. set $U_{\tau,s+1}$ prefix-free.

Substage $t = 0$. Define $\delta_0 = \lambda$. Go to substage 1.

Substage $t+1 \leq s+1$. Given δ_t and for all $\mathcal{S}$-stretgies $\eta \in T$, $F_t(\eta) \in \{s, g, w\}$, the current state of the η-module, define δ_{t+1} as follows:

Case 1. δ_t is an $\mathcal{R}$-strategy. If $s + 1$ is not δ_t-expansionary, then define $\delta_{t+1} = \delta_t{}^\frown\langle 1\rangle$. If $s + 1$ is δ_t-expansionary, and there is no link with top δ_t, then define $\delta_{t+1} = \delta_t{}^\frown\langle 0\rangle$. Otherwise, $s + 1$ is δ_t-expansionary, and there is a link with top δ_t and bottom $\eta \supseteq \delta_t{}^\frown\langle 0\rangle$, define $\delta_{t+1} = \eta$.

Case 2. δ_t is a $\mathcal{P}$-strategy. If δ_t is satisfied, define $\delta_{t+1} = \delta_t{}^\frown\langle 0\rangle$. Otherwise, define $\delta_{t+1} = \delta_t{}^\frown\langle 1\rangle$.

Case 3. δ_t is an $\mathcal{S}$-strategy. Define $\delta_{t+1} = \delta_t{}^\frown\langle F_t(\delta_t)\rangle$.

Let $\xi = \delta_{t+1}$. Say that ξ requires attention at substage $(s + 1, t + 1)$ if ξ has not been satisfied, and:

1. ξ is a $\mathcal{P}$-strategy, and one of the following holds:
   (1a) $x(\xi)$ is not defined, or
   (1b) $\Phi_\xi(x(\xi))[s]\downarrow$.
2. ξ is an $\mathcal{S}$-strategy, and one of the following holds:
   (2a) $\sigma(\xi)$ is not defined, or
   (2b) ξ is ready to open a gap, or
   (2c) ξ is ready to close a gap.

If ξ does not require attention, and $t < s$, then go to substage $t+2$. Otherwise, choose the first one which applies and perform the action as follows. If $t < s$, and ξ opens an ξ-gap, then go to substage $t+2$. Otherwise, define $\delta[s+1] = \delta_{t+1}$, go to stage $s+2$.

Case 1: $\xi = \rho$ is a $\mathcal{P}$-strategy. If (1a) happens, then define $x(\rho)$ as a big number. If (1b) happens, then enumerate $x(\rho)$ into A and say that $\mathcal{P}_\rho$ is satisfied. In any case, initialize all $\zeta > \rho$.

Case 2: $\xi = \eta$ is an $\mathcal{S}$-strategy.

If (2a) happens, define $\sigma(\eta) = 00\cdots01$ with $|\sigma(\eta)|$ big. (Clearly, $\{\sigma(\eta)\} \cup U_{\tau(\eta),s}$ is prefix-free.) Initialize all $\zeta > \eta^\frown\langle w\rangle$.

If (2b) happens (*This action is called* **step 1**), open an η-gap by defining $F(\eta) = g$, and $r(\eta) = 0$. Initialize all $\zeta \geq \eta^\frown\langle w\rangle$. Create a link (τ, η) with top $\tau = \tau(\eta)$, and bottom η.

If (2c) happens (*This action is called* **step 2**), let $v+1 < s+1$ be the stage at which the current η-gap was opened, and let $\sigma = \sigma(\eta)[s]$. Close the η-gap as follows:

step 2a (*Successful Close*) If $\alpha_\eta[s] - \alpha_\eta[v] \geq 2^{-|\sigma(\eta)|}$, then enumerate $\sigma(\eta)$ into $U_{\tau(\eta)}$, and define $F(\eta) = s$. Initialize all $\zeta \geq \eta^\frown\langle g\rangle$. Set $r(\eta) = 0$.

step 2b (*Unsuccessful Close*) If $\alpha_\eta[s] - \alpha_\eta[v] < 2^{-|\sigma(\eta)|}$, then define $F(\eta) = w$ and $r(\eta) = s$ (*to preserve* $\Phi_\eta^A \upharpoonright |\sigma(\eta)|$), cancel $\sigma(\eta)$, initialize all $\zeta \geq \eta^\frown\langle w\rangle$.

This completes the construction.

6 Verification

Let f be the leftmost path. That is, the leftmost $f \in [T]$ such that for all strings $\xi \in T$, if $\xi \leq_{\mathrm{L}} f$ and $\xi \not\subseteq f$, then there are only finitely many stages where $\xi \subseteq \delta[s]$. The following lemmas prove that our construction satisfies all requirements.

(6.1) Lemma Fix n and $\xi \subset f$ with $|\xi| = n$. Then,
(1) $(\exists^\infty s)(\xi \subsetneqq \delta_s)$,
(2) $|G^\xi| = \infty$.
(3) If $|\xi| \equiv 1 \pmod 3$, then ξ receives attention only finitely often.
(4) For any $\tau \subset \xi$, if $|\tau| \equiv 0 \pmod 3$, and $\tau^\frown\langle 0\rangle \subset \xi$, then every link with top τ is removed and there are infinitely many genuine ξ-stages where there are no links with top τ.

(6.2) Lemma Let $\rho \subset \xi$ with $|\rho| \equiv 1 \pmod 3$, and let x be a follower of ρ where $x = \lim_s x(\rho)[s]$, the last follower of ρ appointed at stage s_1 say, where $W_{e,s_1} \cap A_{s_1} = \emptyset$. Then

(1) $(\forall s \in G^\rho)[s > s_1 \to y > R(\rho, s)]$, where $R(\rho, s) = \max\{r(\zeta, s) : \zeta \leq_{\mathrm{L}} \rho\}$.

(2) $\mathcal{P}_{i(\rho)}$ is satisfied.

(6.3) Lemma Suppose that $\tau \subset f$ with $|\tau| \equiv 0 \pmod 3$ and $\tau \mathbin{^\frown} \langle 1 \rangle \subset f$. Then $\alpha_{i(\tau)} \neq \Phi^A_{i(\tau)}$.

(6.4) Lemma Suppose that $\eta \subset f$ with $|\eta| \equiv 2 \pmod 3$.

(1) If $\eta \mathbin{^\frown} \langle s \rangle \subset f$, then $U_{\tau(\eta)} \cap V_{j(\eta)} \neq \emptyset$. Indeed, $\sigma(\eta) \in U_{\tau(\eta)} \cap V_{j(\eta)}$, where $\sigma(\eta)$ is the (unique) string enumerated into $U_{\tau(\eta)}$ when the (last) η-gap is closed (successfully).

(2) If $\eta \mathbin{^\frown} \langle g \rangle \subset f$, then α_η is computable.

(3) If $\eta \mathbin{^\frown} \langle w \rangle \subset f$, then $U_{\tau(\eta)} \cup V_{j(\eta)} \neq \Sigma^*$.

(6.5) Lemma Suppose that $\tau \subset f$ with $|\tau| \equiv 0 \pmod 3$ and $\tau \mathbin{^\frown} \langle 0 \rangle \subset f$. Then U_τ is a presentation of α_τ. Namely, U_τ is prefix-free, and $\sum_{\sigma \in U_\tau} 2^{-|\sigma|} = \alpha_\tau$. Moreover, if α_τ is not computable, then U_τ is also not computable.

This completes the proof of Theorem 6. □

References

1. C. Calude, *Information Theory and Randomness, An Algorithmic Perspective*, Springer-Verlag, Berlin, 1994.
2. C. Calude, P. Hertling, B. Khoussainov, Y. Wang, *Recursively enumerable reals and Chaitin's Ω number*, in STACS '98, Springer Lecture Notes in Computer Science, Vol. 1373 (1988), 596-606.
3. G. J. Chaitin, *Algorithmic Information Theory*, Cambridge University Press, Cambridge, 1987. (Third Print, 1990).
4. R. Downey, D. Hirschfeldt, A. Nies, *Randomness, computability, and density*, to appear.
5. R. Downey, G. LaForte, *Presentations of computably enumerable reals*, Theoretical Computer Science, to appear.
6. A. Kučera, T. Slaman, *Randomness and recursive enumerability*, SIAM Journal on Computing, to appear.
7. M. Li, P. Vitanyi, *Kolmogorov Complexity and Its Applications*, Springer-Verlag, New York, Berlin, Heidelberg, 1993.
8. R. I. Soare, *Recursion theory and Dedekind cuts*, Transactions of American Mathematical Society 140 (1969), 271-294.
9. R. I. Soare, *Recursively Enumerable Sets and Degrees*, Springer-Verlag, Berlin, 1987.
10. E. Spector, *Nicht konstrucktive beweisbare sätze der analysics*, Journal of Symbolic Logic 14 (1949), 145-158.
11. A. M. Turing, *On computable numbers with an application to the Entscheidungsproblem*, Proceedings of American Mathematical Society 42 (1936-7), 230-265; a correction, ibid. 43 (1937), 544-546.

Weakly Computable Real Numbers and Total Computable Real Functions (Extended Abstract)

Robert Rettinger[1], Xizhong Zheng[2,⋆], Romain Gengler[2], and Burchard von Braunmühl[2]

[1] FernUniversität Hagen, 58084-Hagen, Germany
[2] BTU Cottbus, 03044 Cottbus, Germany

Abstract. Let $\mathbf{C}_{sc}$ and $\mathbf{C}_{wc}$ be classes of the semi-computable and weakly computable real numbers, respectively, which are discussed by Weihrauch and Zheng [12]. In this paper we show that both $\mathbf{C}_{sc}$ and $\mathbf{C}_{wc}$ are not closed under the total computable real functions of finite length on some closed interval, although such functions map always a semi-computable real numbers to a weakly computable one. On the other hand, their closures under general total computable real functions are the same and are in fact an algebraic field. This field can also be characterized by the limits of computable sequences of rational numbers with some special converging properties.

1 Introduction

The computability notion of real numbers is introduced by Alan M. Turing in his famous paper [9] on "Turing machine". He called a real number x *computable* if it has a computable binary expansion. Namely, there is a recursive set $A \subseteq \mathbb{N}$ such that $x = x_A := \sum_{i \in A} 2^{-(i+1)}$. Equivalently, x is computable iff there is a computable sequence $(x_n)_{n \in \mathbb{N}}$ of rational numbers which converges to x effectively in the sense that $|x - x_n| \leq 2^{-n}$ for all $n \in \mathbb{N}$. In fact, as pointed by R. M. Robinson [7], corresponding to any classical definition of real numbers there is an equivalent definition of computable real numbers. The class of all computable real numbers is denoted by $\mathbf{C}_e$.

Essentially, we can effectively find an upper bound and a lower bound of any computable $x \in \mathbb{R}$ within any given error. In other words, we can approximate the real number x effectively from both sides, below and above. Sometimes, a real number can only be approximated effectively from one side instead of two. As one of the first such example, Specker [8] has shown that the real number x_A for a nonrecursive r.e. set A is effectively approximable from below but it is not effectively approximable from above. Such kind of real numbers are called *left computable*[1]. Namely, a real number is left computable if it is a limit of

⋆ Corresponding author, zheng@informatik.tu-cottbus.de

[1] Some authors call them recursively enumerable or computably enumerable (see, e.g. [2]).

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 586–595, 2001.
© Springer-Verlag Berlin Heidelberg 2001

some computable increasing sequence of rational numbers. The *right computable real numbers* can be defined accordingly. Left and right computable real number classes are denoted by $\mathbf{C}_{lc}$ and $\mathbf{C}_{rc}$, respectively.

It is not difficult to see that $\mathbf{C}_e$ is closed under the arithmetic operations, namely, it forms an algebraic field. However, the situation for $\mathbf{C}_{lc}$ and $\mathbf{C}_{rc}$ is different. The first easy observation is, that x is left computable iff $-x$ is right computable, and x is computable iff it is both left and right computable. From Specker's example, it follows immediately that the classes $\mathbf{C}_{lc}$ and $\mathbf{C}_{rc}$ are not closed under the arithmetic operations, hence they are not algebraic fields. Moreover, Weihrauch and Zheng [12] have shown that there is a left computable real number x and a right computable real number y such that the sum $x+y$ is neither left computable nor right computable. Therefore even the union $\mathbf{C}_{sc} := \mathbf{C}_{lc} \cup \mathbf{C}_{rc}$ (the class of *semi-computable* real numbers) is not an algebraic field. In other words, the closure of semi-computable real numbers under the arithmetic operations is a proper superset of the semi-computable real number class. This closure is denoted by $\mathbf{C}_{wc}$ and any real number in this closure is called *weakly computable* (w.c., for short). An interesting characterization of w.c. real numbers is (see [12]), that x is weakly computable iff there is a computable sequence $(x_n)_{n\in\mathbb{N}}$ of rational numbers which converges to x *weakly effectively* in the sense that $\sum_{n\in\mathbb{N}} |x_n - x_{n+1}|$ is finite.

As the weakest version of effectiveness on real numbers we can consider simply the limits of any computable sequence of rational numbers without any further restriction but the convergence. We will call a real number x *recursively approximable* (r.a., for short) if there is a computable sequence of rational numbers which converges to x. The class of all r.a. real numbers is denoted by $\mathbf{C}_{ra}$. As shown by Weihrauch and Zheng [12], the class $\mathbf{C}_{wc}$ is a proper subset of $\mathbf{C}_{ra}$ and $\mathbf{C}_{wc}$ is an algebraic field[2].

The class $\mathbf{C}_{ra}$ shares a lot of mathematical properties with $\mathbf{C}_e$. For example, they are all closed under the computable operations (i.e. partial computable real functions) and effectively convergent limits ([13]). On the other hand, the class $\mathbf{C}_{wc}$ does not share these properties any more, as shown by Zheng [13], although it is also an algebraic field. One of the interesting problems remaining open there is, whether $\mathbf{C}_{sc}$ or $\mathbf{C}_{wc}$ is closed under the total computable real functions. A negative answer to this question will be shown in this paper.

During the discussion of closure properties for real number classes under the total computable real functions, it plays an important role, that whether a total computable real function has a finite length on some closed interval[3]. For example, we can easily show that the image $f(x)$ of a semi-computable real number x under a computable real function f of finite length on some closed interval containing x is always weakly computable. But $f(x)$ can be non-weakly computable for general total computable real functions f. Furthermore, we show that the class $\mathbf{C}_{sc}$ is not closed under the total computable real functions, no matter the length of such functions. This holds also for the class $\mathbf{C}_{wc}$. In fact

[2] An earlier but weaker result that $\mathbf{C}_{sc} \subsetneq \mathbf{C}_{ra}$ is proved by Ceĭtin in [3]

[3] The length of the a function means here the length of its graph.

we can even show that $\mathbf{C}_{wc}$ is not closed under the monotone computable total real functions. On the other hand the closures of $\mathbf{C}_{wc}$ and $\mathbf{C}_{lc}$ under general total computable real functions are in fact the same. This common closure is an algebraic field and is strictly between the classes $\mathbf{C}_{wc}$ and $\mathbf{C}_{ra}$. At the end we show that this field can also be characterized by computable sequences of rational numbers with some restriction on their convergence.

2 Preliminaries

In this section we will recall some notions and related known results. Basic notions about classical computability theory are assumed. Since the real numbers can be described by rational numbers, the effectiveness notion on real numbers will be naturally introduced by means of the effectiveness notions on rational numbers. On rational numbers $\mathbb{Q}$, the computability theory is well developed by classical computability theory. For example, the notions of computable function $f : \mathbb{Q} \to \mathbb{Q}$, recursively enumerable set $A \subseteq \mathbb{Q}$, etc. are clear and do not need any further explanation. Especially, we will call a sequence $(x_n)_{n\in\mathbb{N}}$ of rational numbers *computable*, if there is a computable function $f : \mathbb{N} \to \mathbb{Q}$ such that $x_n = f(n)$.

In the last section we have mentioned several classes of real numbers. The relationship among these classes looks like, as shown in [12], the following

$$\mathbf{C}_e = \mathbf{C}_{lc} \cap \mathbf{C}_{rc} \subsetneq \begin{matrix}\mathbf{C}_{lc}\\ \mathbf{C}_{rc}\end{matrix} \subsetneq \mathbf{C}_{sc} = \mathbf{C}_{lc} \cup \mathbf{C}_{rc} \subsetneq \mathbf{C}_{wc} \subsetneq \mathbf{C}_{ra}.$$

For $\mathbf{C}_{wc}$, we have the following equivalent descriptions.

Theorem 1 (Weihrauch and Zheng [12]). *For $x \in \mathbb{R}$, the following are equivalent:*

1. *x is weakly computable;*
2. *There are left computable real numbers $y, z \in \mathbb{R}$ such that $x = y - z$;*
3. *x is in the arithmetic closure of $\mathbf{C}_{lc}$;*
4. *There is a computable sequence $(x_n)_{n\in\mathbb{N}}$ of real numbers which converges to x such that $\sum_{n=0}^{\infty} |x_n - x_{n+1}| \leq c$ for some real number c.*

The computability notion for real numbers can be extended to sequences. A sequence $(x_n)_{n\in\mathbb{N}}$ of real numbers is called *computable*, if there is a computable sequence $(r_{nm})_{n,m\in\mathbb{N}}$ of rational numbers which converges to $(x_n)_{n\in\mathbb{N}}$ uniformly effectively in n and m. Namely, for any $n, m \in \mathbb{N}$, $|r_{nm} - x_n| \leq 2^{-m}$ holds .

Now let us recall the definition of computable real functions. There are a lot of equivalent approaches to define the computable real functions (see e.g. [4,5,6,11]). Here we use the approach of Weihrauch [10,11] by so-called Type-2 Turing machines. A type-2 Turing machine M extends the classical Turing machine in such a way that it accepts infinite sequences as well as finite strings as inputs and outputs. For any input p, $M(p)$ outputs a finite string q if $M(p)$ halts in finite steps with q in its write-only output tape, while $M(p)$ outputs

an infinite sequence q means that $M(p)$ will never halt and keep writing the sequence q on its output tape. A real function[1] $f :\subseteq \mathbb{R} \to \mathbb{R}$ is *computable*, if there is a type-2 Turing machine M which computes f in the sense that, for any $x \in \mathrm{dom}(f)$ and any sequence p (of rational numbers) which converges effectively to x, $M(p)$ outputs a sequence q (of rational numbers) which converges effectively to $f(x)$. It is easy to see that computable function is continuous at any point of its domain.

In this paper we are mainly interested in the closure properties of some real number classes under the computable total real functions. Obviously, the real number x and $x \pm n$, for any $n \in \mathbb{N}$, have completely the same effectiveness in any reasonable sense. Therefore, w.l.o.g., we consider only the real numbers in the interval $[0;1]$ which can always be expressed by x_A for some $A \subseteq \mathbb{N}$. Similarly, we can restrict ourselves to the computable real functions $f : [0;1] \to [0;1]$ for which we have the following equivalent descriptions.

Theorem 2 (cf. Pour-El and Richards [6], Weihrauch [11]). *Let $f : [0;1] \to \mathbb{R}$ be any function. Then the following are equivalent:*

1. *f is computable;*
2. *There is a computable sequence $(p_n)_{n\in\mathbb{N}}$ of rational polygon functions which converges to f uniformly effectively. Namely, $|f(x) - p_n(x)| \leq 2^{-n}$ for any $x \in [0;1]$ and $n \in \mathbb{N}$.*
3. *f is sequentially computable, i.e. f maps any computable sequence $(x_n)_{n\in\mathbb{N}}$ of real number to a computable sequence $(f(x_n))_{n\in\mathbb{N}}$; and f is effectively uniformly continuous, i.e. there is a recursive modulus $e : \mathbb{N} \to \mathbb{N}$ such that $\forall x, y \in [0;1]\, \forall n \in \mathbb{N}\, (|x - y| \leq 2^{-e(n)} \Longrightarrow |f(x) - f(y)| \leq 2^{-n})$.*

Item 2. of Theorem 2 is called Effective Weierstrass Theorem which implies immediately the following lemma where $[0;1]_{\mathbb{Q}} := \{r \in \mathbb{Q} : 0 \leq r \leq 1\}$.

Lemma 1. *Let $f : [0;1] \to \mathbb{R}$ be computable. There is a computable function $\beta : \mathbb{N} \times [0;1]_{\mathbb{Q}} \to \mathbb{Q}$ such that $|f(r) - \beta(n,r)| \leq 2^{-n}$ for any $n \in \mathbb{N}$ and $r \in [0;1]_{\mathbb{Q}}$. In this case we call β an effective determinator of function f.*

Let $\mathbf{C} \subseteq \mathbb{R}$ and $\mathbf{F}$ a set of functions $f :\subseteq [0;1] \to \mathbb{R}$. Denote by $\mathbf{F}(\mathbf{C})$ the image set of $\mathbf{C}$ under the functions of $\mathbf{F}$. Furthermore we define $\mathbf{F}^n(\mathbf{C})$, for any $n \in \mathbb{N}$, inductively by $\mathbf{F}^0(\mathbf{C}) := \mathbf{C}$ and $\mathbf{F}^{n+1}(\mathbf{C}) := \mathbf{F}(\mathbf{F}^n(\mathbf{C}))$. The closure $\mathbf{F}^*(\mathbf{C})$ of $\mathbf{C}$ under the functions of $\mathbf{F}$ is defined now by $\mathbf{F}^*(\mathbf{C}) := \bigcup_{n\in\mathbb{N}} \mathbf{F}^n(\mathbf{C})$. We are especially interested in the classes FIN and TOT of computable total real functions from $[0;1]$ to $[0;1]$ of finite and arbitrary length, respectively. For convenience we will call the function $f \in \mathrm{TOT}$ ($f \in \mathrm{FIN}$) total computable real function (of finite length) without mention of the interval $[0;1]$. Since the composition of two computable real functions is a computable real function, we have $\mathrm{TOT}^*(\mathbf{C}) = \mathrm{TOT}^n(\mathbf{C}) = \mathrm{TOT}(\mathbf{C})$ for any $n \geq 1$. However $\mathrm{FIN}(\mathbf{C}) = \mathrm{FIN}^*(\mathbf{C})$ does not hold in general.

[1] We use $f :\subseteq X \to Y$ to denote that f is a partial function from X to Y while $f : X \to Y$ is a total function from X to Y.

Let $(M_e)_{e\in\mathbb{N}}$ be an effective enumeration of all Turing machines and $(\varphi_e)_{e\in\mathbb{N}}$ an effective enumeration of all computable functions $\varphi_e :\subseteq \mathbb{N} \to \mathbb{N}$ such that φ_e is computed by M_e. We write $\varphi_e(n)\downarrow$ if $n \in \mathrm{dom}(\varphi_e)$ and $\varphi_e(n)\uparrow$, otherwise. The uniformly effective approximation $\varphi_{e,s}$ of φ_e is defined by $\varphi_{e,s}(n) := m$, if $M_e(n)$ halts in s steps and outputs m; and $\varphi_{e,s}(n)$ is undefined, otherwise. Naturally, we assume that $e, n, m \le s$ if $\varphi_{e,s}(n) = m$. For other types of the functions, say, $f :\subseteq \mathbb{N} \to \mathbb{Q}$, a similar effective enumeration and corresponding approximation can be introduced accordingly.

For any alphabet Σ, let Σ^* and Σ^ω be the sets of all finite strings and infinite sequences of Σ, respectively. For $u, v \in \Sigma^*$, denote by uv the concatenation of v after u. If $w \in \Sigma^* \cup \Sigma^\omega$, then $w[n]$ denotes its n-th element. Thus, $w = w[0]w[1]\cdots w[n-1]$, if $|w|$, the length of w, is n, and $w = w[0]w[1]w[2]\cdots$, if $|w| = \infty$. Obviously, $w[n]$ is defined only for $n < |w|$. We will say also that $w[n]$ is undefined (written $w[n] =\uparrow$), if $n \ge |w|$. The unique string of length 0 is always denoted by λ (so-called empty string). For any finite string $w \in \{0;1\}^*$, and number $n \le |w|$, the restriction $w \upharpoonright n$ is defined by $(w \upharpoonright n)[i] := w[i]$ if $i < n$ and $(w \upharpoonright n)[i] :=\uparrow$, otherwise. The generalized restriction $w\lceil n$ is defined by $w\lceil n := w \upharpoonright (n+1)$, if $n < |w|$ and $w\lceil n := w1^{(n+1)-|w|}$, otherwise. For $u, v \in \Sigma^* \cup \Sigma^\omega$, if $u = v \upharpoonright n$ for some $n \le |v|$, then we call u an initial segment of v and denote it by $u \sqsubseteq v$. We denote also $u \ne v$ & $u \sqsubseteq v$ by $u \sqsubset v$. Let $\langle\cdot,\cdot\rangle : \mathbb{N}^2 \to \mathbb{N}$ be a computable pairing function with computable reverse functions.

3 Technical Lemma

In this section we will show two technical results which will be used to construct some total computable real functions satisfying some special conditions. We introduce at first a notion of interval trees.

Definition 1. Let $\delta : \mathbb{N} \to \mathbb{N}^+$ be a function and $\mathbb{I}$ the set of all rational intervals on $[0;1]$ and $\mathbb{N}^*_\delta := \{w \in \mathbb{N}^* : \forall n < |w|\,(w[n] < \delta(n))\}$.

1. A *δ-interval tree* (δ-i.t., for short) on $[0;1]$ is a function $I : \mathbb{N}^*_\delta \to \mathbb{I}$ such that $I(\lambda) = [0;1]$; $\bigcup_{i<\delta(|w|)} I(wi) = I(w)$, for any $w \in \mathbb{N}^*_\delta$; and $\lim_{s\to\infty} l(I(w_s)) = 0$ for any increasing sequence $(w_s)_{s\in\mathbb{N}}$ of $\mathbb{N}^*_\delta$ in the sense of $w_s \sqsubset w_{s+1}$, where $l(I(w))$ is the length of the interval $I(w)$. For any δ-i.t. I and $w \in \mathbb{N}^*_\delta$, the interval $I(w)$ is denoted by $I(w) := [a^\delta_w; b^\delta_w]$ and its length is denoted by $l^\delta_w := b^\delta_w - a^\delta_w$.
2. A δ-i.t. I is called *computable* if the functions $a, b : \mathbb{N}^*_\delta \to [0;1]_\mathbb{Q}$ defined by $a(w) := a^\delta_w$ and $b(w) := b^\delta_w$, respectively, are computable.
3. A δ-i.t. I is called *canonical* if, for any $w \in \mathbb{N}^*_\delta$, the interval $I(w)$ is divided into $\delta(|w|)$ disjunctive subintervals $I(w0), I(w1), \cdots, I(w(\delta(|w|)-1))$ of the same length, namely, $a^\delta_w := \sum_{i<|w|} w[i] \cdot \prod_{j\le i} \delta(j)^{-1}$ and $b^\delta_w := a^\delta_w + \prod_{j<|w|} \delta(j)^{-1}$ where $\sum_{i\in\emptyset}(\cdots) := 0$ and $\prod_{i\in\emptyset}(\cdots) := 1$.

Notice that, for a canonical δ-i.t. I, the set $S^I_w := \{I(wi) : i < \delta(|w|)\}$ is a partition of the interval $I(w)$ and all subintervals in S^I_w have the same length. But, general δ-i.t. I, S^I_w is not necessarily disjunctive. Besides, it is easy to see that a canonical δ-i.t. is always computable, if δ is computable. We are interested mainly in δ-i.t.'s on $[0;1]$. So the phrase "on $[0;1]$" is often omitted later. δ-i.t.'s will be very useful to define some total computable real functions which satisfy special properties, if these properties can be approximated by some "compatible" function ι defined as follows.

Definition 2. Let $\delta_1, \delta_2 : \mathbb{N} \to \mathbb{N}^+$ and $\iota :\subseteq \mathbb{N}^*_{\delta_1} \to \mathbb{N}^*_{\delta_2}$ be functions.

1. ι is called (δ_1, δ_2)-*compatible* if the following conditions hold.
   (a) The domain $\mathrm{dom}(\iota)$ of ι is infinite and is alternate in the sense that, $w(i-1), w(i+1) \notin \mathrm{dom}(\iota)$ for any $wi \in \mathrm{dom}(\iota)$ and $0 < i < \delta_1(|w|)$;
   (b) $\forall w, v \in \mathbb{N}^*_{\delta_1}$ $(w \in \mathrm{dom}(\iota)$ & $v \sqsubseteq w \Longrightarrow v \in \mathrm{dom}(\iota)$ & $\iota(v) \sqsubseteq \iota(w)))$;
   (c) $\forall u, v \in \mathrm{dom}(\iota)\,(|u| = |v| \Longrightarrow |\iota(u)| = |\iota(v)|)$.
2. ι is called (δ_1, δ_2)-*increasing*, if $b^{\delta_2}_{\iota(wi)} \leq a^{\delta_2}_{\iota(wj)}$ for any $w \in \mathrm{dom}(\iota)$ and $i < j$.
3. ι is called *effectively modular* if there is a recursive function $g :\subseteq \mathbb{N} \to \mathbb{N}$ such that, $wi \in \mathrm{dom}(\iota)$ iff $i \in \mathrm{dom}(g)$ & $\iota(wi) = \iota(w)g(i)$, for any $w \in \mathrm{dom}(\iota)$.

Theorem 3. *Let $\delta_1, \delta_2 : \mathbb{N} \to \mathbb{N}^+$ be recursive functions, I_1 a canonical δ_1-i.t., and I_2 a computable δ_2-i.t. such that $l^{\delta_2}_w \leq 2^{-e(|w|)}$ for some recursive function e and all $w \in \mathbb{N}^*_{\delta_2}$. If $\iota :\subseteq \mathbb{N}^*_{\delta_1} \to \mathbb{N}^*_{\delta_2}$ is a (δ_1, δ_2)-compatible computable function, then there is a computable function $f : [0;1] \to [0;1]$ such that $f(I_1(w)) \subseteq I_2(\iota(w))$ for all $w \in \mathrm{dom}(\iota)$. If ι is also (δ_1, δ_2)-increasing, then such computable function f can even be also increasing.*

Notice that, a function f is increasing here means that $x \leq y \Longrightarrow f(x) \leq f(y)$, namely non-strictly increase. The computable function f constructed in Theorem 3 can have finite length if it is increasing. Next Theorem shows that f can have also finite length even if it is not monotone.

Theorem 4. *Let $\delta_1 := k_1$ and $\delta_2 := k_2$ be constant functions, I_1 and I_2 the canonical δ_1-i.t. and δ_2-i.t., respectively. If $2k_2 \geq k_1$ and $\iota :\subseteq \mathbb{N}^*_{\delta_1} \to \mathbb{N}^*_{\delta_2}$ is a (δ_1, δ_2)-compatible and effectively modular function, then there is a computable function $f : [0;1] \to [0;1]$ of finite length such that $f(I_1(w)) \subseteq I_2(\iota(w))$ for all $w \in \mathrm{dom}(\iota)$.*

4 Closure Properties under FIN

In this section we discuss the closure properties of some real number classes under total computable real functions of finite length. We will see that neither $\mathbf{C}_{sc}$ nor $\mathbf{C}_{wc}$ is closed under FIN.

Notice that, the computable function $f : [0;1] \to [0;1]$ defined by $f(x) := 1-x$ has finite length and maps left computable real numbers to right computable ones and vice versa. Furthermore, for any $g \in \mathrm{FIN}$, $g \circ f$ and g have the same

length. Then it is easy to see that $\mathrm{FIN}(\mathbf{C}_{lc}) = \mathrm{FIN}(\mathbf{C}_{rc}) = \mathrm{FIN}(\mathbf{C}_{sc})$. Therefore we often state our results only about the class $\mathrm{FIN}(\mathbf{C}_{sc})$ and omit the other two classes. We show at first that $\mathbf{C}_{sc}$ is not closed under FIN and its image $\mathrm{FIN}(\mathbf{C}_{sc})$ is contained in $\mathbf{C}_{wc}$.

Theorem 5. $\mathbf{C}_{sc} \subsetneq \mathrm{FIN}(\mathbf{C}_{sc}) \subseteq \mathbf{C}_{wc}$.

Proof. (Sketch for the first part) We define a computable function $f : [0;1] \to [0;1]$ by applying Theorem 4. Let $\delta_1(n) = \delta_2(n) = 7$ for any $n \in \mathbb{N}$ and I_1, I_2 the canonical δ_1- and δ_2-interval trees, respectively. Then we have $l_w^{\delta_2} \leq 7^{-|w|}$ for any $w \in \mathbb{N}^*_{\delta_2}$. Define a computable function $\iota :\subseteq \mathbb{N}^*_{\delta_1} \to \mathbb{N}^*_{\delta_1}$ by $\iota(\lambda) := \lambda$ and $\iota(wa) := \iota(w)(6-a)$, if $a \in \{1,3,5\}$, and $\iota(wa) :=\uparrow$, otherwise. Obviously, ι is (δ_1, δ_2)-compatible and effectively modular. By Theorem 4, there is a computable function $f : [0;1] \to [0;1]$ of finite length such that $f(I_1(w)) \subseteq I_2(\iota(w))$ for any $w \in \mathrm{dom}(\iota)$.

Let $(\varphi_i)_{i\in\mathbb{N}}$ be an effective enumeration of computable $\varphi_i :\subseteq \mathbb{N} \to \mathbb{Q}$. Then $H := \{\langle i,n\rangle : \varphi_i(n)\downarrow \ \&\ (\forall\, m < n\,(\varphi_i(m) < \varphi_i(m+1)) \ \text{ or } \ \forall\, m < n\,(\varphi_i(m) > \varphi_i(m+1)))\}$ is an r.e. set. Especially, there is an effective enumeration function $g : \mathbb{N} \to H$ such that

$$g(s) = \langle i,n\rangle \ \&\ n \neq 0 \Longrightarrow \exists s' < s(g(s') = \langle i, n-1\rangle). \tag{1}$$

We construct now a computable increasing sequence $(x_s)_{s\in\mathbb{N}}$ of rational numbers by $x_s := a^{\delta_1}_{w_s}$ for some computable sequence $(w_s)_{s\in\mathbb{N}}$ of $\mathrm{dom}(I_1)$ which is constructed in stages as follows.

Stage $s = 0$: Define simply $w_0 :=$ "1".

Stage $s+1$: Given $w_s \in \mathrm{dom}(I_1)$. Suppose that $g(s) = \langle i,n\rangle$. Define $w_{s+1} := w_s 1$ if $\varphi_i(n) \notin I_2(\iota(w_s\lceil i))$ and, otherwise, define w_{s+1} of length $\max\{i, |w_s|\}+1$ by

$$w_{s+1}[m] := \begin{cases} (w_s\lceil i)[i]+2 & \text{if } \ m = i \ \&\ \varphi_i(n-1) > \varphi_i(n) \\ (w_s\lceil i)[i]+4 & \text{if } \ m = i \ \&\ \varphi_i(n-1) < \varphi_i(n) \\ (w_s\lceil i)[m] & \text{if } \ m < i \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

We can show that $(x_s)_{s\in\mathbb{N}}$ is a well defined computable increasing sequence of rational numbers which converges to x_ω such that $f(x_\omega)$ is not semi-computable.

Next Theorem shows that the class $\mathbf{C}_{wc}$ is not closed under monotone total computable real functions which have, of course, finite length.

Theorem 6. *There is a computable increasing function $f : [0;1] \to [0;1]$ and a weakly computable real number $x \in [0;1]$ such that $f(x)$ is not weakly computable. Hence the class $\mathbf{C}_{wc}$ is not closed under* FIN *and $\mathbf{C}_{wc} \subsetneq \mathrm{FIN}(\mathbf{C}_{wc})$*

Proof. We define at first a recursive function $\theta : \mathbb{N} \to \mathbb{N}$ inductively by $\theta(0) := 5$ and $\theta(n+1) := (\sum_{m\leq n} \theta(m)+1)\cdot(n+1)\cdot 5^{n+1}$. Furthermore, let $\delta_1(n) := \theta(n)$ and $\delta_2(n) := 5$ be two recursive functions, I_1, I_2 the corresponding canonical δ_1-

and δ_2-interval trees, respectively. Then we have $l_w^{\delta_2} \leq 2^{-|w|}$ at first. Define a computable function $\iota : \mathbb{N}^*_{\delta_1} \to \mathbb{N}^*_{\delta_2}$ with $\mathrm{dom}(\iota) = \{1,3\}^*$ by $\iota(w) := w$. Obviously, ι is also (δ_1, δ_2)-compatible and (δ_1, δ_2)-increasing. By Theorem 3, there is an increasing computable function $f : [0;1] \to [0;1]$ such that $f(I_1(w)) \subseteq I_2(\iota(w)) = I_2(w)$. Now we will construct a computable sequence $(x_s)_{s\in\mathbb{N}}$ of rational numbers which converges weakly effectively to some x such that $f(x)$ is not weakly computable. Namely, x satisfies, for all $i \in \mathbb{N}$, the following requirements:

$$R_i : \mathrm{dom}(\varphi_i) = \mathbb{N} \;\&\; \sum_{n=0}^{\infty} |\varphi_i(n) - \varphi_i(n+1)| < i \Longrightarrow \forall^\infty n \, (|f(x) - \varphi_i(n)| \geq 5^{-i})$$

where $(\varphi_i)_{i\in\mathbb{N}}$ is an effective enumeration of computable functions $\varphi_i :\subseteq \mathbb{N} \to \mathbb{Q}$. Let $H := \{\langle i, n\rangle : \forall m \leq n \, (\varphi_i(m) \downarrow)\}$. Then H is r.e. and hence there is a recursive function $g : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{rang}(g) = H$. Especially, we can assume that g satisfies condition (1) in the proof of Theorem 5. The sequence $(x_s)_{s\in\mathbb{N}}$ is defined by $x_s := a^{\delta_1}_{w_s}$ for the computable sequence $(w_s)_{s\in\mathbb{N}}$ of $\mathbb{N}^*_{\delta_1}$ which is constructed effectively in stages as follows.

Stage $s = 0$: Define $w_0 :=$ "1".

Stage $s+1$: Given $w_s \in \mathbb{N}^*_{\delta_1}$. Let $g(s) = \langle i, n\rangle$. If $\sum_{m<n} |\varphi_i(m) - \varphi_i(m+1)| \geq i$ or $\varphi_i(n) \notin I_2(\iota(w_s\lceil i))$, then define simply $w_{s+1} := w_s$ and hence $x_{s+1} := x_s$. Otherwise we define w_{s+1} of length $i+1$ by

$$w_{s+1}[m] := \begin{cases} 1 & \text{if } m = i \;\&\; w_s[m] = 3 \\ 3 & \text{if } m = i \;\&\; w_s[m] = 1 \\ w_s[m] & \text{if } m < i \\ \uparrow & \text{otherwise.} \end{cases} \tag{3}$$

In this case, we call the stage $s+1$ an i-stage.

We can show that there are at most $i \cdot 5^i$ i-stages between stages s_1 and s, for any i and $s_1 < s$ such that s_1 is an i_1-stage for some $i_1 < i$ or $s_1 = 0$ and there is no j-stage between s_1 and s for any $j < i$. Therefore, there are at most $\theta(i)$ i-stages totally. Since $|x_s - x_{s+1}| = \prod_{j\leq i} \delta_1^{-1} \leq \delta_1^{-1}$, the sequence $(x_s)_{s\in\mathbb{N}}$ satisfies $\sum_{n=0}^{\infty} |x_s - x_{s+1}| \leq 2$. Namely, $x := \lim_{s\to\infty} x_s$ exists and is w.c. On the other hand, we can also show that R_i is satisfied for any i and hence $f(x)$ is not weakly computable.

Corollary 1. $\mathrm{FIN}(\mathbf{C}_{sc}) \subsetneq \mathrm{FIN}(\mathbf{C}_{wc})$.

5 Closure Properties under TOT

Different from the case for $\mathrm{FIN}(\mathbf{C}_{sc})$ and $\mathrm{FIN}(\mathbf{C}_{wc})$ we will show in this section that $\mathrm{TOT}(\mathbf{C}_{sc})$ and $\mathrm{TOT}(\mathbf{C}_{wc})$ coincide. Furthermore $\mathrm{TOT}(\mathbf{C}_{wc})$ is an algebraic field which is strictly between $\mathbf{C}_{wc}$ and $\mathbf{C}_{ra}$. This class can also be described by the limits of computable sequences of rational numbers with some kind of restriction on their convergence. Since $\mathrm{TOT}^*(\mathbf{C}) = \mathrm{TOT}(\mathbf{C})$ for any $\mathbf{C} \subseteq \mathbb{R}$ it suffices to mention only the results about $\mathrm{TOT}(\mathbf{C})$.

At first, we introduce a kind of pairing function of real numbers.

Definition 3. *The pairing function* $\oplus : \mathbb{R}^2 \to \mathbb{R}$ *is defined, for infinite* $A, B \subseteq \mathbb{N}$ *by*

$$x_A \oplus x_B := \sum \{5^{-(2i+1)} : i \in A\} \bigcup \{3 \cdot 5^{-(2i+1)} : i \notin A\} \bigcup \{5^{-(2i+2)} : i \in B\} \bigcup \{3 \cdot 5^{-(2i+2)} : i \notin B\}.$$

The pairing function $\oplus$ is unfortunately not computable. However its two inverse functions are computable as shown in the following lemma.

Lemma 2. *The pairing function* $\oplus : \mathbb{R}^2 \to \mathbb{R}$ *satisfies the following conditions:*

1. *If* x *and* y *are left computable, then* $x \oplus y$ *is also left computable;*
2. *There are computable functions* $\pi_1, \pi_2 : [0;1]^2 \to [0;1]$ *such that, for any* $x, y \in [0;1]$, $\pi_1(x \oplus y) = x$, $\pi_2(x \oplus y) = y$ *and* $\pi_1(x) \oplus \pi_2(x) = x$.

Theorem 7. *1.* $\mathbf{C}_{wc} \subseteq \mathrm{TOT}(\mathbf{C}_{lc})$*;*
2. $\mathrm{TOT}(\mathbf{C}_{wc}) = \mathrm{TOT}(\mathbf{C}_{sc})$ *and* $\mathrm{TOT}(\mathbf{C}_{wc})$ *is an algebraic field.*

Now we show that the closure $\mathrm{TOT}(\mathbf{C}_{wc})$ does not exhaust the class $\mathbf{C}_{ra}$.

Theorem 8. $\mathbf{C}_{wc} \subsetneq \mathrm{TOT}(\mathbf{C}_{wc}) \subsetneq \mathbf{C}_{ra}$

Proof. We sketch only the idea for $\mathrm{TOT}(\mathbf{C}_{wc}) = \mathrm{TOT}(\mathbf{C}_{lc}) \subsetneq \mathbf{C}_{ra}$. Let $(\alpha_n)_{n\in\mathbb{N}}$, $(\beta_n)_{n\in\mathbb{N}}$ and $(\gamma_n)_{n\in\mathbb{N}}$ be effective enumerations of computable functions $\alpha_n :\subseteq \mathbb{N} \to \mathbb{N}$, $\beta_n :\subseteq \mathbb{N}\times\mathbb{Q} \to \mathbb{Q}$ and $\gamma_n :\subseteq \mathbb{N} \to \mathbb{Q}$, respectively. For any $a \in \mathrm{TOT}(\mathbf{C}_{lc})$, there is a computable function $f : [0;1] \to \mathbb{R}$ and a computable increasing sequence $(\gamma_k(n))_{n\in\mathbb{N}}$ of rational numbers such that $b := \lim_{n\to\infty} \gamma_k(n) \in [0;1]$ and $f(b) = a$. By Lemma 1, f has a computable determinator function $\beta_j : \mathbb{N} \times \mathbb{Q} \to \mathbb{Q}$. Namely, $|\beta_j(n.r) - f(r)| \leq 2^{-n}$ for any $n \in \mathbb{N}$ and $r \in [0;1]_{\mathbb{Q}}$. Furthermore, by Theorem 2.3, f has a computable modulus $\alpha_i : \mathbb{N} \to \mathbb{N}$ such that $\forall x, y \in [0;1]\, \forall n \in \mathbb{N}\, (|x - y| \leq 2^{-\alpha_i(n)} \Longrightarrow |f(x) - f(y)| < 6^{-n})$.

Therefore, it suffices to construct a computable sequence $(x_s)_{s\in\mathbb{N}}$ of rational numbers such that its limit $x := \lim_{s\to\infty} x_s$ exists and satisfies, for all $e := \langle i, j, k\rangle \in \mathbb{N}$, the following requirements:

$$R_e : \quad \begin{array}{l} \text{If } \alpha_i, \beta_j, \gamma_k \text{ are total and } (\gamma_k(n))_{n\in\mathbb{N}} \text{ converges increasingly to } \gamma_k \\ \text{then } \exists m(|\gamma_k(m) - \gamma_k| \leq 2^{-\alpha_i(m)} \ \&\ |\beta_j(e, \gamma_k(m)) - x| \geq 6^{-(e+1)}). \end{array}$$

To satisfy R_e ($e := \langle i, j, k\rangle$), we choose a rational interval $(a; b)$ of length 6^{-e} as a base interval. This interval is divided by $a := a_0 < a_1 < a_2 < a_3 < a_4 < a_5 < a_6 := b$ into six subintervals I_t for $t < 6$ of the same length $6^{-(e+1)}$. We wait for some stage s such that $\alpha_i(e) := \alpha_{i,s}(e)$ is defined. Then look for the largest $m \leq s$ such that $\gamma_{k,s}(0) \leq \gamma_{k,s}(1) \leq \cdots \leq \gamma_{k,s}(m) \leq 1$ are all defined. If $\beta_{j,s}(e, \gamma_{k.s}(m))$ is also defined and located in the interval I_t for some $t < 6$, then define $x := a_1$, if $t \geq 3$ and $x := a_4$, otherwise. If there is no such s, then define simply $x := a_1$. If at a later stage $s_1 > s$, we find some $m_1 > m$ such that $|\gamma_{k,s_1}(m) - \gamma_{k,s_1}(m_1)| \geq 2^{-\alpha_i(e)}$, then we repeat the above procedure for

m_1 again, and so on. This can happen only finitely often, if $\gamma_k(n)$ converges to some $\gamma_k \in [0;1]$ increasingly. Thus we can always successfully find such x which satisfies R_e.

To satisfy all requirements simultaneously, we apply the above strategies for R_e to an interval $I(w)$ with $|w| = e$ for the canonical δ-interval tree I on $[0;1]$ for $\delta := 6$ by the priority injury technique. The details are omitted here.

At last we characterize the class $\mathrm{TOT}(\mathbf{C}_{wc})$ by some convergence property of computable sequences of rational numbers.

Definition 4. Let $(a_i)_{i\in\mathbb{N}}$ be any sequence of real numbers and $n \in \mathbb{N}$. We define the *degree of n-divergence* of the sequence $(a_i)_{i\in\mathbb{N}}$ as the maximal natural number m which satisfies the condition that there is a chain of natural numbers $i_0 < j_0 \leq i_1 < j_1 \leq \cdots \leq i_m < j_m$ such that $|a_{i_s} - a_{j_s}| \geq 2^{-n}$ for any $s \leq m$.

Theorem 9. *The real number x belongs to $TOT(\mathbf{C}_{wc})$ iff there is a computable sequence $(a_i)_{i\in\mathbb{N}}$ of rational numbers which converges to x and a recursive function $f : \mathbb{N} \to \mathbb{N}$ such that, for all $n \in \mathbb{N}$, $\alpha(n) \leq f(n)$, where $\alpha(n)$ is the degree of n-divergence of $(a_i)_{i\in\mathbb{N}}$.*

References

1. K. Ambos-Spies, K. Weihrauch and X. Zheng. Weakly computable real numbers. *J. of Complexity.* 16(2000), 676–690.
2. C. Calude, P. Hertling, B. Khoussainov, and Y. Wang. Recursive enumerable reals and Chaintin's Ω-number, in *STACS'98*, pp596–606.
3. G. S. Ceĭtin A pesudofundamental sequence that is not equivalent to a monotone one (Russian), *Zap. Naučn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. (LOMI)*, 20(1971), 263–271.
4. A. Grzegorczyk. On the definitions of recursive real continuous functions, *Fund. Math.* 44(1957), 61–71.
5. Ker-I Ko. *Complexity Theory of Real Functions*, Birkhäuser, Berlin, 1991.
6. M. Pour-El & J. Richards. *Computability in Analysis and Physics.* Springer-Verlag, Berlin, Heidelberg, 1989.
7. R. M. Robinson. Review of "R. Peter: 'Rekursive Funktionen', Akad. Kiado. Budapest, 1951," *J. of Symbolic Logic*, 16(1951), 280.
8. E. Specker. Nicht konstruktiv beweisbare Sätze der Analysis, *J. Symbolic Logic* 14(1949), 145–158
9. A. M. Turing. On computable number, with an application to the "Entscheidungsproblem". *Proceeding of the London Mathematical Society*, 43(1936), no.2, 230–265.
10. K. Weihrauch. *An Introduction to Computability.* EATCS Monographs on Theoretical Computer Science Vol. 9, Springer-Verlag, Berlin, Heidelberg, 1987.
11. K. Weihrauch. *Computable Analysis.* Texts in Theoretical Computer Science, Springer-Verlag, Heidelberg 2000.
12. K. Weihrauch & X. Zheng. A finite hierarchy of the recursively enumerable real numbers, *MFCS'98* Brno, Czech Republic, August 1998, pp798–806.
13. X. Zheng. The closure properties on real numbers under limits and computable operations. *COCOON'2000*, Sydney, Australia, July 26-28, 2000. pp170–179.

Turing Computability of a Nonlinear Schrödinger Propagator

Klaus Weihrauch[1] and Ning Zhong[2]

[1] Theoretische Informatik I, FernUniversität, 58084 Hagen, Germany
Klaus.Weihrauch@FernUni-Hagen.de
[2] Clermont College of the University of Cincinnati, Batavia, OH 45103-1785, USA
Ning.Zhong@uc.edu

Abstract. We study Turing computability of the nonlinear solution operator S of the Cauchy problem for the Schrödinger equation of the form

$$i\frac{du}{dt} = -\frac{d^2u}{dx^2} + mu + |u|^2u$$

in $\mathbb{R}$. We prove that S is a computable operator from $H^1(\mathbb{R})$ to $C(\mathbb{R};H^1(\mathbb{R}))$ with respect to the canonical representations.

There is a rich mathematical theory for the Schrödinger equation of the form

$$i\frac{du}{dt} = (-\triangle + m)u + |u|^2u, \quad t \in \mathbb{R},\ x \in \mathbb{R}^n,$$

where $\triangle$ is the Laplace operator in $\mathbb{R}^n$ and m is a real constant. The equation is widely used in several domains of applied physics. For example, it can be considered as the classical approximation to the field equation for a quantum mechanical nonrelativistic many body system with a two body δ-function interaction. It is also used in the Landau-Ginsburg theory of superconductivity.

Our interest is to study computability of the solution operator. It is commonly believed among physicists that for processes which can be described by well-established theories, such as quantum systems described by the Schrödinger equation, the future behavior can be computed with arbitrary precision at least in principle from sufficiently precisely given initial conditions, where the computations can be performed on digital computers, hence on Turing machines. In this note we put the common belief into a formal basis of computable analysis, which is the theory of real number computation. We study computability of the nonlinear Schrödinger propagator within the framework of Type-2 Theory of Effectivity (TTE) [Wei00], which is based on work by Turing, Grzegorczyk, Lacombe, Ko, Kreitz and Weihrauch, Pour-El and Richards, and many others [Tur36, Grz57, KW85, PER89, Ko91]. For simplicity of notation we consider only the case of space dimension $n = 1$.

The main result is the following theorem.

J. Wang (Ed.): COCOON 2001, LNCS 2108, pp. 596–599, 2001.
© Springer-Verlag Berlin Heidelberg 2001

Theorem 1 *The solution operator $S : H^1(\mathbb{R}) \to C(\mathbb{R}; H^1(\mathbb{R}))$ of the Cauchy problem*

$$i\frac{du}{dt} = (-\triangle + m)u + |u|^2 u, \quad t \in \mathbb{R}, \quad x \in \mathbb{R}, \tag{1}$$

$$u(0)(x) = \varphi(x), \quad \varphi \in H^1(\mathbb{R}), \tag{2}$$

where $S(\varphi) = u$ and m is a computable real number, is $(\delta_{H^1}, [\rho \to \delta_{H^1}])$-computable.

We use the framework of TTE[Wei00]. Let $L^2(\mathbb{R})$ denote the set of all L^2-functions defined on $\mathbb{R}$ equipped with L^2-norm, $H^s(\mathbb{R})$ the Sobolev space, $s \geq 0$, $C(\mathbb{R}, H^s(\mathbb{R}))$ the set of all continuous functions $g : \mathbb{R} \to H^s(\mathbb{R})$ equipped with the compact-open topology, and $\mathcal{S}(\mathbb{R})$ the Schwartz space defined by

$$\mathcal{S}(\mathbb{R}) = \{\phi \in C^\infty(\mathbb{R}) : \forall \alpha, \beta \in \mathbb{N}, \sup_{x \in \mathbb{R}} |x^\alpha \phi^{(\beta)}(x)| < \infty\}.$$

Let δ_{L^2} and δ_{H^s} be the standard representations of $L^2(\mathbb{R})$ and $H^s(\mathbb{R})$, respectively, $[\rho \to \delta_{H^s}]$ the representation of $C(\mathbb{R}, H^s(\mathbb{R}))$, and δ_{SC} the Cauchy representation of $\mathcal{S}(\mathbb{R})$. For the details about these representations the reader is refereed to [Wei00], [WZ99], and [WZ01]. We need another representation, denoted as $\tilde{\delta}_{H^s}$, of $H^s(\mathbb{R})$ [WZ01]. The representation $\tilde{\delta}_{H^s}$ makes use of the classical result that the set of Schwartz functions is dense in $H^s(\mathbb{R})$.

Lemma 2 *The representation δ_{H^s} is equivalent to the representation $\tilde{\delta}_{H^s}$, if $s \in \mathbb{R}$ is computable.*

Proof: See [WZ01].

We now consider the Cauchy problem (1, 2). We use a fixed point argument to construct a Type-2 Turing machine which computes a $[\rho \to \delta_{H^1}]$-name of the solution when input a δ_{H^1}-name of the initial function. The Cauchy problem $(1, 2)$ can be recasted in the form of the integral equation

$$u(t) = U(t - t_0, \psi) - i \int_{t_0}^{t} U(t - \tau, |u(\tau)|^2 u(\tau)) d\tau \tag{3}$$

with $u(t_0) = \psi$. We will first construct a segment of u on the interval $[t_0 - T_\varphi, t_0 + T_\varphi]$ by a fixed point iteration, where T_φ is a constant depending only on the initial condition φ of (1, 2). The following two conservation laws are needed.

Lemma 3 *If $u : \mathbb{R} \to H^1(\mathbb{R})$ is a solution of (1), then for any $t, t', t_0 \in \mathbb{R}$,*

$$||u(t)||_{H^1} \leq f(||u(t_0)||_{H^1}) \tag{4}$$

$$||U(t - t', u(t'))||_{H^1} \leq f(||u(t_0)||_{H^1}), \tag{5}$$

where $f(x) = x \cdot (1 + 4\sqrt{2}x)^{1/2}$ and $U : \mathbb{R} \times \mathcal{S}(\mathbb{R}) \to \mathcal{S}(\mathbb{R})$ is the free evolution defined by $U(t, \psi) := (2\pi)^{-1/2} \int_{\mathbb{R}} e^{i\xi x - i|\xi|^2 t - imt} \hat{\psi} d\xi$.

Proof: See [WZ01].

We define two maps A and G. The map $A : \mathbb{R} \times H^1(\mathbb{R}) \times C(\mathbb{R}, H^1(\mathbb{R})) \to C(\mathbb{R}, H^1(\mathbb{R}))$ is defined by

$$A(t_0, \psi)(v)(t) = U(t - t_0, \psi) - i \int_{t_0}^{t} U(t - \tau, |v(\tau)|^2 v(\tau)) d\tau,$$

and the map $G : \mathbb{R} \times C(\mathbb{R}, H^1(\mathbb{R})) \to C(\mathbb{R}, H^1(\mathbb{R}))$ is defined by

$$G(t_0, v)(t) = -i \int_{t_0}^{t} U(t - \tau, |v(\tau)|^2 v(\tau)) d\tau.$$

On a small neighborhood I of t_0, $A(t_0, \psi)$ is a contraction on $C(I, H^1(\mathbb{R}))$, if ψ is not too big.

Lemma 4 *Let r_φ be a rational number such that $r_\varphi - 1 \leq ||\varphi||_{H^1} \leq r_\varphi$, let R_φ be the least integer upper bound of $f(r_\varphi) + 1$ and let $I := [t_0 - T_\varphi, t_0 + T_\varphi]$, where $T_\varphi = 1/(32R_\varphi^2)$. Then for any $\psi \in H^1(\mathbb{R})$ such that $||\psi||_{H^1} \leq R_\varphi$ and any $v_1, v_2 \in C(\mathbb{R}; H^1(\mathbb{R}))$,*

$$||A(t_0, \psi)v_1 - A(t_0, \psi)v_2||_I = ||G(t_0, v_1) - G(t_0, v_2)||_I \leq \frac{1}{2}||v_1 - v_2||_I \quad (6)$$

where $||v||_I := ||v||_{C(I, H^1(\mathbb{R})} = \sup_{t \in I} ||v(t)||_{H^1(\mathbb{R})}$.

Proof: See [GV79].

The fixed point of the contraction $A(t_0, \psi)$ is the solution of (3) such that $u(t_0) = \psi$. We are able to show that the restriction of the operator A to $\mathcal{S}(\mathbb{R})$ is computable (see Lemma 5 below). So we will approximate $\psi \in H^1(\mathbb{R})$ by a Cauchy sequence ψ_k of $\mathcal{S}(\mathbb{R})$-functions (see Lemma 2) and use approximations of the fixed points of $A(t_0, \psi_k)$ to compute the fixed point of $A(t_0, \psi)$ (see Lemma 6 below).

Lemma 5 *The restriction B of the operator A to $\mathcal{S}(\mathbb{R})$ is*

$$(\rho, \delta_{SC}, [\rho \to \delta_{SC}], [\rho \to \delta_{SC}])\text{-computable.}$$

Proof: See [WZ01].

Lemma 6 *Let $||\psi||_{H^1} \leq R_\varphi$ and $||\psi - \psi_k||_{H^1} \leq 2^{-k}$ for $k = 0, 1, \ldots$, where ψ_k's are $\mathcal{S}(\mathbb{R})$-functions. Let v_ψ be the fixed point of $A(t_0, \psi)$. Then there are computable functions $g_1, g_2 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that*

$$||v_\psi - \left(A(t_0, \psi_{g_2(R_\varphi, m)})\right)^{g_1(R_\varphi, m)}(0)||_I \leq 2^{-m}$$

Proof: See [WZ01].

Combining lemmas 4 – 6 we can construct a Type-Two Turing machine which computes $u(t)$ from the initial condition $\varphi \in H^1(\mathbb{R})$ and arbitrary $t \in \mathbb{R}$.

Proof of Theorem 1: See [WZ01].

References

[Grz57] Andrzej Grzegorczyk. On the definitions of computable real continuous functions. *Fundamenta Mathematicae*, 44:61–71, 1957.

[GV79] J. Ginibre and G. Velo. On a class of nonlinear Schrödinger equations. i. the Cauchy problem, general case. *Journal of Functional Analysis*, 32(1):1–32, 1979.

[Ko91] Ker-I Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.

[KW85] Christoph Kreitz and Klaus Weihrauch. Theory of representations. *Theoretical Computer Science*, 38:35–53, 1985.

[PER89] Marian B. Pour-El and J. Ian Richards. *Computability in Analysis and Physics*. Perspectives in Mathematical Logic. Springer, Berlin, 1989.

[Tur36] Alan M. Turing. On computable numbers, with an application to the "Entscheidungsproblem". *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

[WZ99] Klaus Weihrauch and Ning Zhong. The wave propagator is Turing computable. *Lecture Notes in Computer Science*, Vol.1644, 697–706, 1999.

[ZW00] Ning Zhong and Klaus Weihrauch. Computability theory of generalized functions. Informatik Berichte 276, FernUniversität Hagen, Hagen, September 2000.

[WZ01] Klaus Weihrauch and Ning Zhong. Turing computability of a nonlinear Schrödinger propagator. Informatik Berichte, FernUniversität Hagen, to appear.

Author Index

Ahn, Hee-Kap 237
Aichholzer, Oswin 101
Alber, Jochen 318
Arslan, Abdullah N. 111
Aurenhammer, Franz 101

Bach, Eric 473
Bespamyatnikh, Sergei 121
Bläser, Markus 1
Bose, Prosenjit 142
Braunmühl, Burchard von 586

Caballero-Gil, Pino 257
Cai, Jin-Yi 473
Cai, Maocheng 247
Castanho, Carla Denise 59
Chan, Wun-Tat 432
Chandran, L. Sunil 308, 384
Chang, Ee-Chien 453
Chen, Danny Z. 150
Chen, Guangting 519
Chen, Qingfeng 561
Chen, Wei 59
Chen, Zhixiang 121, 540
Cheng, Siu-Wing 237
Cheong, Otfried 159, 237
Chrobak, Marek 483
Chwa, Kyung-Yong 443
Cohen, Myra B. 420
Colbourn, Charles J. 420

Dai, H.K. 228
Dang, Zhe 529
Dawande, Milind 409
Deng, Xiaotie 247
Díaz, Josep 298
Du, Ding-Zhu 375, 509
Duncan, Rob 191

Eğecioğlu, Ömer 111
Epstein, Leah 463

Fang, Qizhi 247
Feng, Haodi 414
Fernau, Henning 318
Finocchi, Irene 357, 368
Froncek, Dalibor 420
Fujiwara, Akihiro 59

Gąsieniec, Leszek 483
Gengler, Romain 586
Golin, Mordecai 237
Gu, Xun 86

Heath, Samuel A. 64
Hemaspaandra, Lane A. 49
Hempel, Harald 49
Hernández-Goya, Candelaria 257
Hsu, Wen-Lian 207
Hu, Xiaobo (Sharon) 150
Hurtado, Ferran 101

Ibarra, Oscar H. 529
Igarashi, Yoshihide 571
Iwama, Kazuo 28

Kang, Liying 375
Kemmerer, Richard A. 529
Kikuchi, Yosuke 400
Kim, Hyun-Sung 262
Kim, Jae-Hoon 443
King, Valerie 268
Koucký, Michal 11
Krasser, Hannes 101

Lam, Tak-Wah 432
Li, Xiang-Yang 91, 181
Lou, Dingjun 349
Lu, Chin Lung 338
Luo, Xudong 561

Madhavan, C.E. Veni 328
Morin, Pat 142
Motegi, Kazuhiro 571
Motoki, Mitsuo 502

Nakano, Shin-ichi 131
Narayanaswamy, N.S. 328
Nardelli, Enrico 278
Nickelsen, Arfst 49
Niedermeier, Rolf 318
Nishizeki, Takao 288

Obokata, Kumiko 571
Okabe, Yasuo 28
Okamoto, Kyoichi 493
Omori, Michiko 571
Oostrum, René van 237

Petersen, Holger 218
Petreschi, Rossella 368
Preparata, Franco P. 64
Proietti, Guido 278

Qian, Jianbo 191
Qiao, Hong 375

Rettinger, Robert 586
Rytter, Wojciech 483

Serna, Maria 298
Shan, Erfang 375
Shibata, Yukio 400
Shin, Chan-Su 159
Siebert, Bodo 75
Stewart, Iain A. 39
Su, Kaile 561
Su, Zheng-Yao 338
Subramani, K. 550
Sung, Ting-Yi 390

Takase, Toshiro 28
Tan, Xuehou 170, 201
Tang, Chuan Yi 338
Thilikos, Dimitrios M. 298
Thorup, Mikkel 268
Ting, Hing-Fung 432
Tran, Nicholas 21

Vigneron, Antoine 142, 159

Wada, Koichi 59
Wang, Huaiqing 561
Wang, Jeng-Jung 390
Wang, Kanliang 121
Wang, Lusheng 509
Wang, Yu 181
Watanabe, Osamu 493
Weihrauch, Klaus 596
Widmayer, Peter 278
Wong, Wai-Ha 432
Wu, Guohua 576
Wu, Shiquan 86
Wu, Xiaodong 150

Xu, Baogang 395, 509
Xue, Guoliang 519

Yap, Chee 453
Yoo, Kee-Young 262

Zhang, Chengqi 561
Zhang, Shichao 561
Zheng, Xizhong 586
Zhong, Ning (Cincinnati) 596
Zhong, Ning (Guangzhou) 349
Zhou, Xiao 288
Zhu, Binhai 121, 191
Zhu, Shanfeng 247